AFRL-IF-RS-TR-2005-49
Final Technical Report
February 2005

THE GENESIS OF CYBERSCIENCE AND ITS MATHEMATICAL MODELS (CYBERSCIENCE)

SRI International, System Design Laboratory

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J799

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-49 has been reviewed and is approved for publication

APPROVED: /s/

NANCY A. ROBERTS
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Advanced Computing Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2005	Final Mar 00 – Feb 04

4. TITLE AND SUBTITLE
THE GENESIS OF CYBERSCIENCE AND ITS MATHEMATICAL MODELS (CYBERSCIENCE)

5. FUNDING NUMBERS
C - F30602-00-C-0087
PE - 62301E
PR - IAST
TA - 00
WU - 01

6. AUTHOR(S)
Steven Dawson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SRI International, System Design Laboratory
333 Ravenswood Avenue
Menlo Park California 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-49

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Nancy R. Roberts/IFTB/(315) 330-3566/ Nancy.Roberts@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The Cyberscience project has developed a framework for an integrated approach to secure systems development called security co-design. Acknowledging the need to integrate security into the development process from the beginning, but recognizing that security and functionality are different in character, security co-design separates development into security and functional tracks that strongly influence each other. The security co-design methodology aims to account for all critical aspects of development, including requirements capture, implementation, and the construction of an information assurance case (IAC). By analogy to safety cases, an IAC seeks to establish that the security requirements of the system are met, and to identify specific points of failure to be addressed if certain requirements are not met. The development of a methodology and tool support for the construction of IACs has been the primary focus of the Cyberscience project. This report documents the security co-design methodology, the principles and goals of IAC development, an exploration of tool support for IAC construction, and an examination of possible alternative approaches.

14. SUBJECT TERMS
Computer Security, Secure Systems, Systems Development, Information Assurance, Information Assurance Case

15. NUMBER OF PAGES
82

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1 Introduction

The serious study of security in computer systems has existed for nearly as long as computer systems themselves. From the beginning it has been recognized that security is difficult to get right, even for systems that, by today's standards, would be considered reasonably self-contained and isolated from other systems. Today, we are faced with both the growing difficulty of information security and the critical importance that it be done well. Information systems are growing ever larger, and their interactions with other systems are growing more complex. They also face increased exposure to vulnerable infrastructure and attacks from increasingly sophisticated adversaries. Simultaneously, the world is becoming increasingly dependent on information systems in all aspects of human endeavor, and improving their security is vital. Almost daily there are new reports of security breaches on systems at all levels, from personal computers used for e-mail, shopping, and personal finance, to the large e-business servers through which a growing portion of the world's business is conducted. Increased attention to security is needed not only to stem the more immediate and short-term damage that results from these breaches, but also to preserve and build confidence in the information infrastructure.

Despite broad acknowledgment that information security is a growing problem, and one no longer confined to the largest and traditionally most security-conscious organizations, there appears to be little agreement on how the problem should be addressed. However, there is general agreement, at least within the security research community, on characteristics of approaches that do not work and on those that can be expected to be more successful. For example, it has been demonstrated repeatedly that security is difficult to retrofit to systems not designed with security in mind. While security technologies such as firewalls, cryptography, and anti-virus software clearly help, they invariably leave vulnerabilities to be exploited, as demonstrated by the continuing success of new attacks against ubiquitous

PC software such as Outlook™ and super-user daemons in Unix. Furthermore, security cannot be considered in isolation — interactions with the environment and other systems must be taken into account [CMS01] — as the recent concern over *cross-site scripting* vulnerabilities in Web servers amply demonstrates. As is now widely argued [Neu95, Olt01], security must be treated as a *systems* issue if it is to be successful. Security must be an integral part of system development.

To begin to address the needs of secure systems development, the Cyberscience project[1] at SRI has developed a framework for an integrated approach to secure systems development called *security co-design*. Acknowledging the need to integrate security into the development process from the beginning, but recognizing that security and functionality are different in character, the security co-design approach separates the development effort into security and functional tracks that strongly influence each other.

Needs of Secure Systems Development Broadly speaking, any approach to secure systems development needs to address three main issues:

1. *Requirements*: getting the security requirements right and expressing them in a way that is useful to the rest of the development process

2. *Implementation*: building the system to meet its security requirements and maintaining compliance with the requirements throughout the life cycle of the system.

3. *Information assurance (IA) case*: making a convincing, auditable, maintainable case that the implemented system does (or does not) meet its security requirements.

[1] As a historical note, the name "Cyberscience" was chosen for this project as originally proposed to DARPA ATO for the development of a mathematical theory and related set of tools for reasoning about cyberspace phenomena. The project was subsequently moved to the OASIS program in DARPA IPTO and refocused on research in the development of secure systems, but the original project name was retained.

The importance of determining the actual security requirements of a system cannot be overstated. In an analysis of computer security problems [Neu95], Neumann identifies requirements failure as a contributing deleterious cause in all but two specific events and in at least some events in all general categories of security failures. In addition, a recent symposium devoted to the topic of security requirements engineering [Pur01] is indicative of the increased attention in the security community to the requirements issue. The implementation issue, of course, covers all implementation-related phases of system development: design, implementation, testing, deployment, and maintenance. Security requirements must be accounted for in all phases.

A perhaps less obvious, but no less important, issue is the construction of an information assurance case (IAC). By analogy to safety cases [BS93, WKM97, BB98], an IAC seeks to establish convincingly that the security requirements of the realized system are met, and to identify specific points of failure to be addressed in the event that certain requirements are not met. The IAC can be used by system developers in maintaining information assurance throughout the lifetime of the system, by system assessors to minimize potential liability risk, by certification teams who must certify system compliance with relevant regulations, and by management to ensure acceptable implementation and maintenance costs.

The development of a methodology and tool support for the construction and maintenance of IACs has been the primary focus of the Cyberscience project. This report documents the Cyberscience team's progress on this development, including the security co-design methodology itself, the principles underlying the concept of an IAC, the goals of IAC development, and an exploration of tool support for IAC construction and maintenance, including both an initial Cyberscience-developed approach and an examination of possible alternative approaches.

2 Security Co-design

The main thrust of the Cyberscience project has been the development of a framework and methodology for development of secure systems. We call our approach *security co-design*, by analogy with hardware/software co-design [BR95, De 94, FP91, KL94, KAJW93] and influenced by relevant work in dependable software architectures [GRS99, HDRS99, MXR95, MQRG97, Rie99] and the development of safety-critical systems [Lev95, MOD96, IEC95]. One basic tenet of the security co-design approach is that security considerations are as fundamental to system development as functional considerations and must therefore be integrated into the development process from the beginning. Another is that the skills and expertise required for security work are not the same as those required for functional development, and it is not reasonable to expect or require all members of the development team to be experts on how to achieve both safety and security. These observations lead to one of the key elements of the co-design approach: security and functional development follow separate (but not independent) tracks with strong mutual influence. The goal of this form of separation is to allow talent to be focused where it is most effective, while ensuring that security concerns are properly accounted for in the functional development and vice versa.

For any integrated approach to the development of secure systems to be successful, it must include significant tool support to make its application feasible. If, instead, the approach were simply to mandate the use of additional techniques and methodologies that must be applied manually, then the cost of development would become prohibitive, the effort would be scaled back, or important elements of the approach would be omitted. For example, consider the canonical approach for the procurement of high-assurance systems, governed for many years by the DoD's so-called Orange Book [dod85] (or the corresponding European requirements), and more recently by the Common Criteria. Depending on the level of security required, systems developed to these standards must be subjected to rigorous

development processes, analysis, testing, and evaluation by an accepted certification authority. While the resulting system is almost certainly more secure (by some measure) than one developed *ad hoc*, the specialized skills, expertise, and training required to carry out the development and certification make the cost of such systems prohibitive to all but the most security conscious environments, such as military and intelligence organizations.

In contrast, our co-design vision is intended to expand the scope and availability of information assurance techniques by laying the groundwork for tools to assist in the development process and by allowing developers to adapt their use of the tools to the desired level of system security. The idea is to *augment*, rather than replace, the engineering processes already being used by system developers. We argue that security co-design will be most successful and incur the least overhead when the members of the team are able to carry on with their work with minimal interference and without significantly onerous additional or new procedures. This is not to say that the benefits of security co-design are without cost. The envisioned interaction between the security and functional development teams does incur overhead; clearly, any additional work implies additional cost. (Of course, substantial savings can also result from earlier detection of security flaws.) The goal is to keep the additional net cost reasonable relative to the desired level of assurance.

2.1 Rationale

The Cyberscience project has pursued specific elements under the same broad theme established in decades of computer security research [Lev95, Neu95, Olt01]: security must be an integral part of systems development if it is to be successful. At the same time, we recognize that the skills and expertise required for effective security work are in many ways different from those required for effective systems work (from a functional standpoint). These differences in required skills and expertise undoubtedly arise in part from a funda-

mental distinction between the character of functionality and that of security: functionality is concerned with what a system *must* do, while security is concerned with what a system *must not* do. To take an oversimplified view, we might say that a systems engineer, who is charged with the task of building a system that meets certain functional requirements, is most concerned with finding ways to construct a system that does all the things it needs to do; the resulting system may in fact do more. A security expert, on the other hand, is most concerned with ensuring that the system does not do, directly or indirectly, anything it should not do. To the security expert, anything the system does beyond its functional requirements could be another potential avenue for security breaches. In short, the skills and expertise of a systems engineer are geared more toward making things work, while those of the security expert are geared toward keeping things protected.

This is not to argue that there is no overlapping of concerns between the functional and security realms. If an effort to design and build a secure system is to succeed, clearly each side must have substantial awareness and understanding of the other's requirements. One way to accomplish this would be to assemble a team whose members are well-versed in both the functional and security aspects of systems development, but this is unrealistic, since individuals with significant talent and training on both sides are rare, relative to the pool of available talent for the two sides separately. Nor would it be realistic or cost-effective to train the team in all the required areas of expertise. For these reasons, we argue that a better approach is to consider the functional and security aspects of system development as separate tracks, but with strong mutual influence. The notion of separation is important, because it explicitly focuses the different required skill sets where they are most useful. From the standpoint of strong mutual influence, there needs to be continual communication between the two sides to ensure that the requirements of each are accounted for by the other as the effort proceeds.

Influence from the security side to the functional side involves the security team informing

designers and implementors of the security requirements and helping them to make design and implementation decisions that lead to the satisfaction of those requirements. During the system development process, designers and implementors face many choices, such as hardware platforms, use of off-the-shelf software, implementation languages, and algorithms. The range of choices may of course be constrained by the functional requirements themselves, as well as cost considerations, and the skills, experience, and preferences of the development team, among other factors. Security requirements, too, can be viewed as constraints on the design and implementation options available to the development team. For example, a security requirement stating that the system must protect information from unintended disclosure would most likely eliminate the use of cleartext communication over public network links as an implementation option. A primary responsibility of the security side of the co-design approach is to make sure that the security constraints are considered when design and implementation decisions are made. This may include the identification of additional design and implementation alternatives that meet the security requirements.

Influence from the functional side to the security side involves the designers and implementors keeping the security team informed of functional requirements and design and implementation choices, as this information drives the development of more specific security requirements. For example, at the architectural level, it may be possible to satisfy the functional requirements of a particular information system either through a central mainframe with directly connected terminals or through a distributed system involving networked workstations. From the security team's point of view, it may very well be possible to adopt either architecture in building a system that satisfies the top-level security requirements. However, at a more detailed level, the security constraints entailed by a mainframe-based system will most likely be quite different from those of a distributed system. Is it therefore critical for the security team to know what design and implementation alternatives are being considered so that they can make progress on elaborating the security

requirements, which, in turn, feed back into the functional track.

2.2 A Framework for Security Co-design

Based on the particular needs of secure systems development outlined earlier and the co-design philosophy just described, we have developed a framework and methodology for security co-design that addresses all the major needs in an integrated way. This framework is strongly influenced by existing approaches to systems engineering, in particular, our work on dependable software architectures [GRS99, HDRS99, MXR95, MQRG97, Rie99] and design of safety-critical systems [Lev95, Neu95]. A major motivating goal of the framework is that it be *practical* and have a *sound mathematical basis*. On the practical side, the methodology aims to augment, rather than replace, existing developmental practices. As suggested by the earlier discussion of our co-design philosophy, we believe that the maximum benefit to secure systems development will be obtained by allowing team members to apply their existing skills and expertise appropriately, rather than insisting that members be retrained or otherwise forced to adopt fundamentally new procedures. At the same time, the emphasis on mutual influence between the security and functional development tracks should naturally lead to enhancement of the skills and knowledge of both sides. From the standpoint of providing a sound mathematical basis for secure systems development, the methodology incorporates accepted and well-understood formal approaches to security and system modeling.

Figure 1 illustrates the security co-design methodology at a high level. Our approach can be characterized generally (and somewhat ideally) as a top-down development process, although it explicitly allows feedback to, and revisiting of, earlier developmental stages. The process starts with a statement of top-level mission objectives, which may or may not even refer to a computer or information system *per se*, but which should capture what is

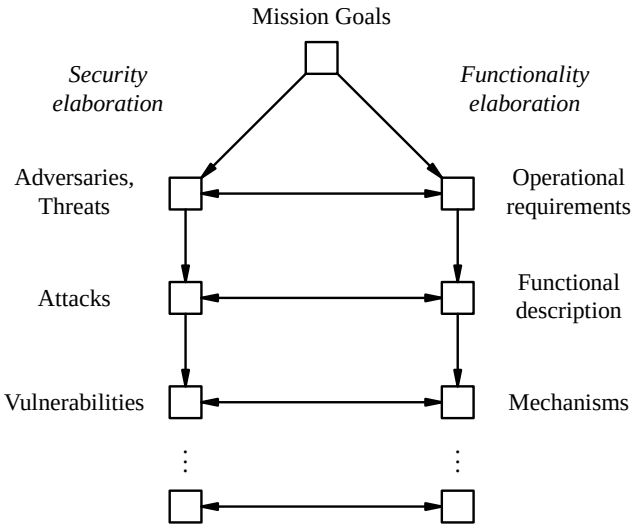

Figure 1: Security co-design method

to be achieved, by whom, and in what environment. That is, the mission objectives should include not only top-level functional requirements, but also assumptions about the target user community, the intended operational environment, and cost constraints. The mission objectives will also include a statement of top-level security requirements, which will generally address (at least) the following major categories of security issues: confidentiality, integrity, and availability.

- The process starts from top-level mission goals, environmental assumptions, cost constraints.
- Iterative elaboration is performed at successively more concrete levels of abstraction.
- Security requirements constrain design/implementation options.
- Different options must be evaluated with respect to security and functional requirements, exposing tradeoffs and forcing conscious, documented, and justifiable choices throughout the development.

- Security and functional elaboration steps are (or can be) interleaved; there is no requirement that there be one security elaboration step for every functional elaboration step (or vice versa).

By following the elaboration/evaluation process, all the desired products (or at least all the raw material for all the desired products) are generated: requirements, architecture, design, implementation, and evidence for the IAC. That is, everything is recorded as the process proceeds. We do not necessarily insist that all the documents be generated and kept up to date during the process, but a co-design "record" should be maintained with all the required information.

Main Elements of the Framework The basic approach to security co-design is intended to provide answers to two questions:

1. *How should security requirements by expressed?*
2. *How should an information assurance case be constructed?*

Our goal in answering these questions is to provide a more scientific foundation for security engineering. Thus, we have sought to answer the first question by providing a formal language for expressing and reasoning about security requirements, and we have sought to answer the second by providing a formal, yet easily understandable, notation for expressing the argument that security requirements are satisfied. The security co-design method provides the system engineering framework in which these notations can be effectively employed.

The general approach of separate-but-coordinated elaboration of functional and security aspects follows directly from adequacy criteria for answers to the two questions. In particular, we believe that we should not require extensive changes in existing system development

methods. Experienced developers of secure systems are rightly wary of any proposed radical changes in development methods that have been proven in practice to "work", in the sense that the resulting systems may not fully meet all requirements, but do prove to be reasonably secure in practice. This constraint suggests that the formal analysis of security concerns can best be integrated into existing development practice by having a team of security analysts provide proactive and reactive advice to the developers as development proceeds. This, in turn, suggests that a complete design record be captured during development, so that security analysts will have all the information they need to do their jobs without having to require the functional developers to supply the information to them. Of course, the idea of maintaining a complete development record has other advantages, and has frequently been proposed in the past as a solution to various software engineering problems, but the benefit of having the record seems especially compelling when an IAC must be constructed, because just about any design decision may have a serious impact on security.

Security Requirements Elaboration One of the most difficult issues in attempting to develop a widely useful approach to dealing with security requirements in system development is the lack of consensus on just what security requirements *are*. Even the basic concepts employed in security requirements statements vary considerably. Security requirements can be expressed in terms of

- potential system vulnerabilities that must be avoided,
- attacks, or general attack strategies, that must not succeed,
- system threads, or use cases, that must never occur,
- capabilities of adversaries that must be insufficient to allow them to interfere with system operation,

or in any of a host of different ways. While there are logical relations among these concepts — for example, attacks exploit vulnerabilities, so eliminating a certain vulnerability guarantees that attacks essentially relying on exploitation of that vulnerability will fail — emphasis on any one to the exclusion of the others entails commitment to single security perspective, which may not be appropriate in all circumstances. This lack of consensus is one reason why there is no standard methodology for security requirements analysis, and helps explain why security requirements are often left implicit, or are only partially and informally captured. In fact, requirements that certain mechanisms intended to provide security — passwords, encryption, firewalls, and so forth — be used often substitute for true security requirements, although this leaves the question of what security properties these mechanisms are intended to provide unanswered and, hence, makes evaluation of whether use of the mechanisms has the intended effect impossible. Some approaches to security even entirely eschew the notion of requirements, opting instead for requiring the use of system development methods that tend to result in more secure systems but that do not guarantee any particular system security properties.

We believe that all these different concepts have a role to play in capturing security requirements. The different concepts correspond to different levels of abstraction in the system's functional design. They play roughly the same role that *styles* [Gar96] play in functional description. For example, it makes sense to talk about system attack scripts at the functional level where the external interface to the system has been defined, but talk of a potential vulnerability in a particular component — say, an overflow of some buffer — makes sense only at a lower level where the system component has been introduced in the design. That the attack must fail is a constraint on the lower-level design; that vulnerabilities such as undetected buffer overflow must be avoided is an elaboration of the attack constraint, one way of making sure it is satisfied. Whatever mechanism is introduced to ensure that buffer overflow cannot occur also guarantees that the attack will fail.

A major goal of the Cyberscience project is to put the process of developing system security requirements on a more scientific footing.

- The process must provide effective guidance to the requirements developer, and whether a particular set of security requirements was developed in accordance with the process must be determinable by independent reviewers, just as in the case of the current method-oriented approach.
- The result of the process, the requirements statement, must provide effective guidance to the system developers.
- There must be a scientific justification that the process results in the right security requirements, i.e., all the requirements that must be satisfied if the system is to perform its mission.

Information Assurance (IA) Case Construction Since complete statements of security requirements, even in informal terms, are rare, complete arguments that a system satisfies its security requirements are rare as well. Often, a considerable body of evidence that a system has desirable security properties is collected — ranging from formal verifications of protocols employed to red team failures to breach — but the evidence is left to speak for itself, rather than used as premises in an argument that the system is sufficiently secure to perform its mission. As a result, the strength of the evidence is hard to assess, and omission of evidence required to assure high confidence in the system's security is easily overlooked. Thus, there is a sharp contrast between security cases for systems with stringent security requirements and safety cases for systems with stringent safety requirements. Safety cases specify exactly what evidence is relevant, what safety hypotheses are influenced by that evidence, and how strongly the evidence influences the safety hypotheses. The other main goal of our effort is to show how to bring a level of rigor to construction of an IAC that

is comparable to the level of rigor commonly seen in safety cases. This will place IAC on a more scientific basis, allowing for review and refinement of the case when more information is obtained, when the system or its security requirements change, and so on. Thus, much higher levels of confidence that security requirements are satisfied can be obtained.

3 Information Assurance Cases

In current practice, many different approaches are employed in the attempt to achieve an adequate level of security.

- There is intensive analysis of the system design, where analysis techniques range from informal inspection of system design documents to formal analysis of mathematical models of the system.
- Standard technology — such as firewalls, encryption, and intrusion detection — that provides security-related functionality is incorporated in the design.
- Best state-of-practice software engineering techniques — such as extensive testing — are employed in development.
- Systems are continually patched to eliminate flaws that are discovered as a result of attacks by red teams and, after deployment, by actual opponents.

This list could be lengthened indefinitely, but two main points are already evident. First, there is typically a great deal of evidence collected to support a claim that a system is adequately secure. Second, the evidence is as disparate as it is voluminous.

Given all the effort devoted to ensuring security, how is it that systems so often fail to meet their security goals? We believe that one factor is the gap between what is directly established by all the evidence that the system is secure and security objectives. For example, if we have evidence that a standard encryption algorithm was used to protect the information in a certain file, and we also have evidence that the encryption algorithm was correctly implemented, then we have good evidence that the information in the file cannot be accessed without the decryption key. However, we are still far from having established the confidentiality of the information in the file. Even leaving aside the obvious possibility of an

attacker obtaining a key, there are many other ways the information might be obtained, for example, insertion of a "trojan horse" that supplies the information to an attacker prior to encryption or after decryption into the storage management system.

These considerations suggest that establishing a desired high-level security property from the available evidence is bound to require a complex argument. Such arguments are, at best, incompletely recorded during the design process. For example, the design record might include a comment that a certain security feature was included in order to thwart a certain class of attacks, or that an analysis of the code shows that a certain class of attacks will fail. However, such arguments are generally incomplete: they rest on assumptions that have not been made explicit, and it is often questionable whether the cited evidence actually provides adequate support for the conclusion. Moreover, the conclusions purportedly established are typically much lower-level than the real security objectives, which have to do with system availability and integrity, information confidentiality, and other high-level properties, rather than failure of some class of attacks.

Ultimately, determination of whether the high-level security objectives have been satisfied is left to the judgment of experts. Typically, the reasoning process these experts employ to arrive at their evaluations is entirely ephemeral, which has a number of disadvantages.

- The reasoning is *non-reviewable*. No one else can check the reasoning for gaps or errors.
- The reasoning process is *non-repeatable*. Even if another expert reaches the same conclusion, there is no way of determining whether or not he reached it by the same route. Thus the reasoning process itself can be validated only indirectly.
- As a result, the reasoning process is *non-improvable*. There is no way to determine whether a given pattern of reasoning can be relied upon generally to provide correct results, since the details of the expert's reasoning are unknown.

- And, most important, the reasoning is *non-maintainable*. Systems evolve after deployment, and there is no way of determining whether or how any given change to the system should influence the belief that it is adequately secure.

At best, experts' assessment processes are codified in standards that offer recipes for secure system development. There is no question that this offers certain advantages. For example, if systems that satisfy a standard frequently prove inadequately secure, an attempt can be made to improve the standard. However, because the argument that satisfaction of the standard can be expected to guarantee adequate security has not been made explicit, determining what has gone wrong can be problematic. This may be one of the reasons why the standards-based approach has not been very successful in yielding high levels of security. Another reason may be that the variation in systems and security needs is too great for a simple cookbook-style approach to satisfactorily address.

It should also be noted that independent assessment of a system's security is an expensive business. Detailed knowledge of the system, its security requirements, and all the evidence relevant to determining whether those requirements have been satisfied is needed. Only members of the development team have this knowledge, but they are hardly likely to find any security holes that they have allowed to slip through during design and development. If an independent assessment is to be based on more than a rather shallow understanding of the system and the security needs, the assessor must spend weeks — more likely, months — "getting up to speed", that is, learning everything about the system that is relevant to the assessment. This impact on cost and schedule is usually unacceptable. Hence, independent assessment, including any assessment performed as part of a formal system certification process, is generally based on a relatively shallow understanding of the system.

We hypothesize that a solution to these problems is to make the argument that the system satisfies its security objectives explicit and complete. One reason for thinking so is an

analogy that can be drawn between safety and security. Systems with stringent safety requirements must be certified safe prior to deployment. The certification process essentially amounts to review of a *safety case*, an explicit argument by the system's developers that the safety requirements have been satisfied. Only if the argument is found to be convincing is the system is certified. The argument can be checked for gaps and errors. If gaps are discovered, they can be filled. If errors are discovered, they can be fixed. If a form of reasoning employed on the argument does not reliably lead to correct results, conclusions established via its use can be discounted by the certifiers. If changes are made to the system, their impact on the safety case can be evaluated. Collecting all the relevant evidence and making its relevance explicit makes it relatively easy for system certifiers to "get up to speed".

We have tested this hypothesis by constructing partial *information assurance cases* (IACs) for two prototype systems having stringent information assurance requirements.

3.1 Safety Cases

In the safety world, the provision of a safety case for each procured critical system is the norm and is frequently mandated by certification authorities. Safety cases are required for military systems, the offshore oil industry, rail transportation, and the nuclear industry.

A safety case is an explicit argument that a system satisfies relevant safety requirements. Typical safety requirements state that the probability of certain system failure modes or certain effects of system failure (e.g., loss of human life due to system failure) must be less than some small parameter. For some systems and parameter values, the expected number of failures that will be observed during the lifetime of the system is large enough that system failure statistics will reveal whether safety requirements have been satisfied; in other cases, failure will be so infrequent that whether requirements have been satisfied will

always remain a matter of conjecture.

Prior to fielding the system, it is generally impossible to *ensure* that safety requirements have been satisfied. But it is possible to develop arguments that support reasonably high confidence that safety requirements have been satisfied, and that is the objective in safety case creation.

The safety case can be broken down into three parts.

- First, there is a collection of relevant *evidence*. The evidence consists of all the facts relevant to the safety assessment. This can range from relatively hard data, such as system test results, to quite soft data, such as adherence to certain system engineering practices during development. In particular, both the results of system analyses — e.g., computations of failure rates based on statistical simulation — and any evidence supporting the assumptions upon which system analyses are based are included.

- Second, there is a collection of *safety requirements*. These include not only the safety requirements given as part of the system specification, but also any derived safety requirements extracted from the given requirements during the analysis process. For example, any categorization of system failure modes introduces additional, more specific, safety requirements. Also, all the assumptions made during safety requirements and system analysis are included among the requirements, since evidence must be provided to show that these assumptions are satisfied.

- Finally, there is a collection of *arguments* that provide the grounds for believing that the safety requirements are supported given the evidence. These arguments can be specific to the particular evidence, more general (i.e., have the form *generally, evidence of this sort indicates that a goal of this sort is satisfied*). Arguments may or may not explicitly indicate the strength of support provided by evidence, and may or

may not determine the net strength of support given the entire body of evidence.[2]

A good introduction to the best current state-of-practice in safety case development can be obtained by examining the strengths and weaknesses of one of the premier commercial tools for safety case definition, the Adelard Safety Case Editor (ASCE).[3] ASCE represents the safety case as a directed multigraph, with three node types and three link types. The three node types — *evidence*, *claims*, and *arguments* — correspond quite directly to the three parts of the safety case identified above. The first link type, *is evidence for*, indicates that some evidence node provides evidence for some argument node, claim node, or other evidence node. (The latter two cases are probably best thought of as resulting from the elision of a trivial argument node.) The second link type, *supports*, links argument nodes to claim nodes that it supports. These *supports* links come in various strengths, indicating whether the argument provides weak, strong, or some intermediate degree of support for the claim. The third link type, *is a subclaim of*, is used to indicate decomposition of a claim node into subclaim nodes. Like *supports* links, *is a subclaim of* links come in various strengths, to indicate the importance of truth of the subclaim to truth of the claim.

ASCE's main selling point is that it makes the structure of an informal safety case explicit, and does so in a somewhat more useful way than the tables of claims and their support representative of typical industrial practice. For example, by associating strengths with *is a subclaim of* links and *supports* links, the ASCE representation suggests where strengthening arguments will have the greatest impact on the overall strength of the safety case: strengthening weak support links to important subclaims. The arguments themselves remain informal, which has some substantial practical advantages. First, providing wildly

[2]Unfortunately, this terminology has not been standardized. For example, in the Safety Argument Manager (SAM), a computer-based tool for constructing safety cases, the analogues of claims are called "goals" and the analogues of arguments are called "warrants". However, the concepts are essentially the same.

[3]A trial copy can be downloaded from the Adelard Web site, `www.adelard.com`.

disparate sorts of evidence for an argument is not a problem; there is no issue of how some piece of evidence can be represented in a particular reasoning framework. Second, no particular technical skill is required to understand the safety case for a system, just enough familiarity with the domain and the system to allow the descriptions of the evidence, argument, and claims to be understood.

However, informal arguments have some inherent shortcomings as well. The most important of these is the absence of a normative standard that determines whether, given the evidence, the arguments do in fact strongly support the safety claims. In technical terms, one would say that there is no criterion for determining whether the argument is *inductively strong*[4] — or, if the conclusions have been appropriately qualified and the correctness of the relevant principles of non-demonstrative inference has been included among the assumptions of the argument, *deductively valid*. Thus, substantial domain expertise is required to determine whether the evidence really provides adequate support for the safety claims. Providing explicit arguments certainly makes this decision easier; however, there is still a possibility that experts will disagree in their assessments of the strength of a safety case. When disagreements occur, the absence of an objective standard that determines who is correct means that the disagreement amounts to a difference of opinion and there is no mechanism to help achieve convergence.

Thus, one substantial research focus has been the formalization of safety cases. The best known work in this area is the attempt to represent safety cases using Bayesian networks [DMS95, LW97]. Roughly, the idea is to replace an informal argument that connects evidence E to safety claim H by $P(E \mid H)$ and $P(E \mid \sim H)$, the probability of observing E given that H is true and the probability of observing E given that H is false. Similarly,

[4]This terminology is a bit unfortunate, since it suggests that induction is the only relevant principle of non-demonstrative inference. (When the expression gained currency, this was generally thought to be the case.)

an informal argument that subclaim H' supports claim H is replaced by $P(H' \mid H)$ and $P(H' \mid \sim H)$. These conditional probabilities, together with the conditional independence assumptions implicit in the structure of the network, determine a probability distribution. In particular, the probability of each safety claim H in the system requirements given the totality of evidence $\mathcal{E}$,

$$P\left(H \;\middle|\; \bigwedge_{E \in \mathcal{E}} E\right),$$

can be calculated via repeated application of Bayes's Theorem.

The principal advantage of this formalization is that it solves the problem of evaluating the inductive strength of the safety case: given the evidence and the probability distribution, the probabilities of the safety claims are uniquely determined. If the case for some safety claim is strong, according to this criterion, but seems weak to some expert assessor, then either

(1) the assessor must disagree with some specific estimate of a conditional probability or some specific independence assumption,

(2) the expert's assessment is based on evidence not explicit in the safety case, or

(3) his assessment is based upon flawed reasoning.

If the assessor can defend his own differing estimate of conditional probabilities and conditional independencies, or can present additional evidence, the accuracy safety case can be improved. And so there is a method for focusing on specifics in a disagreement among experts that improves the chances of achieving consensus.

But the advantages of formalization are not unalloyed. Experience shows that experts find it difficult to estimate the relevant conditional probabilities, and have little faith that their estimates are accurate to even one significant figure. The Bayesian calculations thus produce results much less reliable than the use of precise numeric probabilities suggests. For

example, one would naturally be inclined to prefer a system design supported by a safety case where all required safety properties have a probability of 0.97 or greater to a competing design supported by a safety case where some required safety properties have a probability of only 0.9. However, given the unreliability of typical conditional probability estimates, the difference between a computed probability of 0.9 and 0.97 is not significant, that is, the difference provides no basis for preferring what appears, prima facie, to be the safer design.

Our belief is that, in practice, an intermediate degree of formalization is preferable to either extreme. Replacing conditional probabilities with a small finite range of values — perhaps a scale of 1 to 5, with 1 representing *very unlikely*, 2 representing *somewhat unlikely*, 3 representing *as likely as not*, 4 representing *somewhat likely*, and 5 representing *very likely* — greatly simplifies the problem of obtaining parameter value estimates from an expert. Similarly, Bayesian updating can be replaced by having the expert provide a function for computing a likelihood value for a claim from the likelihood values of its supporting evidence or subclaims. In practice, experts seem to find very simple functions sufficient, often just minimum, maximum, and average. Given the analogy between safety cases and IA cases, which the following discussion addresses in more detail, we also believe that an intermediate degree of formalization is most appropriate for IA cases as well.

3.2 SEAS as an IAC tool

The analogy between safety cases and IACs is based on the notion that IACs should play the same role in security assessment that safety cases do in safety assessment. Given the intended similarity in function, considerable similarity in structure is natural. Thus, we assume that an IAC consists of evidence, IA claims — some of which are given as IA requirements, and some of which are derived by analysis —, and explicit arguments linking

evidence to claims.[5] Given the experience of the safety community, we decided that a graph-based approach is clearly preferable to a linear text-based approach. As mentioned in the previous section, we think a level of formality greater than the informal hypertext of ASCE, but less than the probability distributions characterized by Bayesian nets, is the best compromise between practicality and the attractions of theory.

As a starting point for our IA case development approach, we chose the Structured Evidential Argumentation System (SEAS), developed by the Artificial Intelligence Center at SRI.[6] SEAS was originally developed to aid intelligence analysts in assessing evidence that either supports or refutes hypotheses, with the overall goal of anticipating potential crises around the world. The system was motivated in part by the observation that, while formal methods are difficult to apply to problems of intelligence analysis, decision makers could benefit from an intuitive, easy-to-use system that provides structure, rigor, and automation. Thus, although SEAS was not designed with our application in mind, it nevertheless shares our goal of combining broad usability with scientific rigor. More important, SEAS incorporates several specific features that make it a good match to our needs:

[5] Attempts to exploit an analogy between safety and security have often been criticized on the grounds that safety is inherently probabilistic — one never claims that failure is impossible, merely very infrequent — where security is not. However, two responses are possible. First, while high-level security claims may not be probabilistic, IACs are not intended to establish that the claims are *definitely* true, merely that they are *probably* true, where "probable" is being used in the subjectivists' (a.k.a. Bayesian) sense rather than the frequentists' sense. Second, since we know, based on experience, that all complex systems fail to provide perfect confidentiality, integrity, and so on, it arguably makes good sense to replace these absolute concepts by more probabilistic notions. For example, given that no real system can guarantee integrity in all circumstances, a more reasonable requirement is that failure of integrity is merely very infrequent. In fact, Littlewood [LW97] has proposed *expected effort to breach*, an explicitly probabilistic notion, as a practical replacement for a range of traditional security properties.

[6] For more information about SEAS, see `http://www.ai.sri.com/~seas`.

- It is *graph based*.
- It fully formalizes non-demonstrative arguments; that is, conclusions have strengths based on strengths of hypotheses and strengths of influence.
- It replaces probabilities by small ranges of discrete values that are meaningful to both developers and consumers.
- It replaces conditional probabilities and Bayesian inference by simple user-selected rules for strength propagation (e.g., *min*, *max*, *mean*).

SEAS is built upon a foundation of mature, widely used software to support various aspects of evidential reasoning developed over the past two decades at the AI Center. For example, both the Grasper and Gister systems are components of SEAS.[7]

SEAS is a Web-based system that supports the creation and exploitation of a "corporate memory" organized around three main object types: *argument templates*, which are hierarchically structured sets of interrelated questions; *arguments*, which are instantiations of argument templates with answers to the questions relative to particular situations; and *situation descriptors*, which characterize the situations to which the argument templates apply. In the context of IA case development, argument templates are the generic structures of hypotheses pertaining to particular IA goals, arguments are instantiations of these arguments with evidence, and situation descriptors characterize the part(s) of the system to which the argument templates apply. In SEAS terminology, arguments are indexed by situations, which in the IA context, means that arguments pertaining to particular parts of the system can be readily identified and retrieved.

While developing the prototype IACs using SEAS, we investigated the utility of various extensions to the system. One of our longer-term goals is to enable formal analyses of IACs,

[7]For more information on Grasper and Gister, see `http://www.ai.sri.com/software_list`.

for example, a formal analysis of the breadth of coverage of the IAC relative to the system design, usage scenarios, and other relevant factors. Doing so, without compromising ease of use, is by no means straightforward.

3.3 IAC Construction and Maintenance

The construction of an IAC is not an after-the-fact activity. To be as complete and convincing as possible, development of an IAC should be initiated at the earliest phases of system development and maintained throughout the system life cycle. It can be exceedingly difficult to recapture in the latter stages of a system engineering effort all the analysis, reasoning, and decision-making that went into the process, which in turn is likely to make IAC construction after the fact more expensive and the resulting IAC less compelling.

The fact is that much of the evidence and argumentation needed for a convincing IAC is generated, at least implicitly, during a typical system engineering effort. For example, system architects make design decisions intended to satisfy the functional and security requirements of the system, subject to cost and schedule constraints. Each such decision has some justification, and this justification can become part of an argument that a system that faithfully implements the design will satisfy its requirements. Similarly, implementors select or produce hardware and software components intended to realize the design faithfully. Here, too, the selections have some justification, while the components have certain evaluable characteristics (e.g., by analysis, simulation, testing, and debugging) that can provide evidence to support claims of correct implementation. Similar observations apply to system deployment and maintenance activities.

One of the keys to developing convincing IACs is to make all the argumentation and evidence explicit and to capture it completely during all phases of the system life cycle. Indeed, this notion is one of the primary motivations for the use of a kind of corporate

memory we call the *co-design object base* as a fundamental part of our security co-design methodology. Acknowledging the need to integrate security into the system development process from the beginning, but recognizing that security and functionality are different in character, the security co-design approach separates the development effort into security and functional tracks that strongly influence each other. The co-design object base, or COB, is a central component to support this methodology. The COB is essentially a living history that records the evolution of a system's development. It constitutes a central store from which can be generated all the critical products of a secure-system development effort, in particular, the IAC. In fact, IAC construction is the driving application for COB development. And, as with the IAC itself, we are using SEAS as the starting point for COB development, because of its existing support for maintenance of a corporate memory.

A better understanding of the contents of an IAC can be gained by examining examples. The design-stage IACs for two different systems are described in the next section.

4 Test Cases for the IAC Concept

The use of IACs cannot be explored in depth without focusing on concrete examples. To better grasp the possible structure and composition of an IAC, we developed a design-level IAC in outline form for a dependable intrusion-tolerant Web server (the *DIT system*) developed under a separate project at the SRI System Design Laboratory (SDL) [VAC+01].

In addition, we have used SEAS to develop a design-level IAC for an intrusion-tolerant version of a Joint Battle Infosphere system (IT-JBI) that is being developed under DARPA sponsorship by a team (including SDL) led by BBN Technologies.

The choice of systems used to test the IAC concept and SEAS is not critical to our discussion, and we do not address in detail here the need for the DIT and IT-JBI systems nor their use. However, the fact that the systems were under development at the time of IAC construction was helpful, since, as mentioned above, some of the information designers use to verify dependability of a system is often lost, unavailable, or out-of-date after the system is complete. Also, because the goal of an intrusion-tolerant system is to obtain higher dependability from lower-dependability components, the argument that the system meets its requirements is particularly important and nontrivial.

4.1 An IAC for DIT

The goal of the DIT system is to provide, at reasonable cost, a system for high-availability distribution of Web content, by incorporating widely available, relatively low-assurance COTS software into a high-assurance intrusion tolerant design. The emphasis is on availability and integrity, not confidentiality, of the service.

The system is based on the observation that widely available COTS Web server software is feature-filled and complex, and tends to contain security vulnerabilities. (Examples in-

clude the infamous Code Red virus, which attacked Microsoft IIS under Windows, and the recent "Slapper" Linux worm that exploits Apache/OpenSSL.) But since different Web server programs and operating systems typically have different vulnerabilities, a system with redundant diverse Web servers on diverse platforms may be able to provide a greater assurance of availability and integrity, provided we have reliable mechanisms to compare and forward responses from the redundant servers to clients. The DIT system is a network of redundant COTS servers and other machines that provide such mechanisms.

The architecture (Figure 2) contains a *proxy* to forward client requests to a collection of diverse *application servers* running COTS software and a monitoring subsystem that helps contain intrusions. The proxy is a hardened platform running a small amount of custom code. The simplicity and customized nature of the software on the proxy makes the proxy more amenable to hardening than the application servers, which are running more complex, harder-to-verify COTS software. The proxy accepts client requests, forwards them to a number of application servers, compares the content returned by the application servers, and, assuming enough agree, sends the corroborated answer back to the client. The proxy and application servers communicate over a private network that is monitored by an intrusion detection system (IDS). The IDS provides assurance that ill-behaving compromised application servers will be detected and corrected (e.g., by rebooting from read-only media), so that compromises are likely to remain limited to a small number of application servers. An *agreement policy* determines which and how many servers are queried by the proxy for each client request, and how sufficient agreement is determined.

Since we are describing an IAC for a partially complete system, we will only sketch the scope of the IAC and the material it will contain, and then describe some portions of the case in more depth.

At the highest level, the design-level IAC is an aggregation of several types of evidence and arguments, including

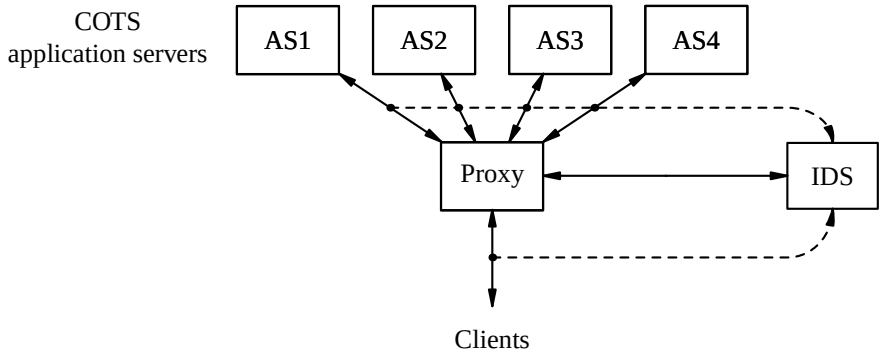

Figure 2: Architecture of the DIT system.

- a *design assurance argument* that the system is correctly designed, i.e., that the system will meet its requirements making certain assumptions about behavior of off-the-shelf components, and
- *component reliability* assessments for off-the-shelf components that are included in the system.

The design assurance argument also contains assessments of the tools used to build the system. The main types of components in DIT are COTS Web server software, operating systems, computer hardware, and network hardware. For each component, we collect evidence that it will behave as needed in the design, relying on component specifications, past performance of the component, and reliability of the component provider.

The design argument makes assumptions about components and interactions of components that are verified in the component assessment and testing portions of the IAC. But testing also provides some redundant evidence, duplicated by the design argument, that the system will operate properly. Note that at this level — as at lower levels — assertions can be combined in more than one way. They can combine in a deductive fashion, in which new assertions are inferred from old ones, or in an aggregative way, in which similar, inde-

pendently derived assertions join to give greater assurance.

Since evidence for the latter two portions of the IAC is incomplete, we now focus on the first portion, the design assurance argument.

A design assurance argument is an assembly of evidence, design details, and reasoning that makes a convincing case that the design of the system, from abstract architecture to the most concrete details of implementation and operation, meets appropriate operational and security requirements. It is important that the argument include descriptions that are as concrete as possible, since the point is to argue that the system itself runs according to its higher-level requirements. For instance, formal verification focuses on the relationship between two descriptions, such as between a specification and a piece of code. An argument would include this verification, but should also give some reason, not necessarily formal, that the system actually executes the code that was verified — and not, for instance, Trojan code.

The argument can be viewed as an explicit representation of the type of argument designers implicitly create when developing a system: it is a structured assembly of all the design decisions, reasoning, and factual information that would be used to explain each phase of system design. Typically, at any given point, a developer focuses on part of the system, at a certain level of abstraction. Properly expressed, the requirements, design choices, reasoning, and assumptions being made at that moment can be used as a "design element" that forms a part of the assurance argument.

The approach we use to capture the assurance argument for the DIT system is to assemble many such design elements, linking them together by checking assumptions and requirements (Figure 3). At the top, we have the requirements for the design: high-availability Web service at reasonable cost with a certain throughput capability. The design element below it describes the topology and basic function of the system: one proxy that commu-

nicates with the client and forwards requests and responses between the client and each of multiple COTS application servers. From this design, it is clear that in order to meet the requirement that expected attacks be tolerated, we must make a number of assumptions, including

- A majority of the application servers are functioning properly at any one time.
- The proxy implements an agreement policy that serves correct (majority) content.
- With high assurance, the proxy is not compromised by attacks.

Design at more concrete levels naturally focuses on the three basic parts of the architecture: the proxy, the application servers, and the network components connecting them.[8] Design elements for each of these parts detail the more concrete internal design of the components and how it meets the architecture assumptions.

We also look more concretely at how the components fit together. For example, we specify the exact protocols used between proxy and application servers (HTTP over TCP/IP on Ethernet). We also argue that the concrete interoperation follows the assumptions of the more abstract architecture design. For instance, once we know that the application servers are computers running COTS operating systems and software, and that they are connected by Ethernet, we realize the possibility of a single compromised application server attacking and compromising additional application servers. We argue that this is unlikely, provided we assume that traffic between application servers is restricted and that compromised application servers will be detected and rebooted. We progress to the monitoring mechanisms, including the intrusion detection system [PN97, NP99], to verify these assumptions.

[8] In this case, each design element seems to focus on a physical entity, but this is not the case in general. The elements simply reflect natural or convenient points of view for the designers.

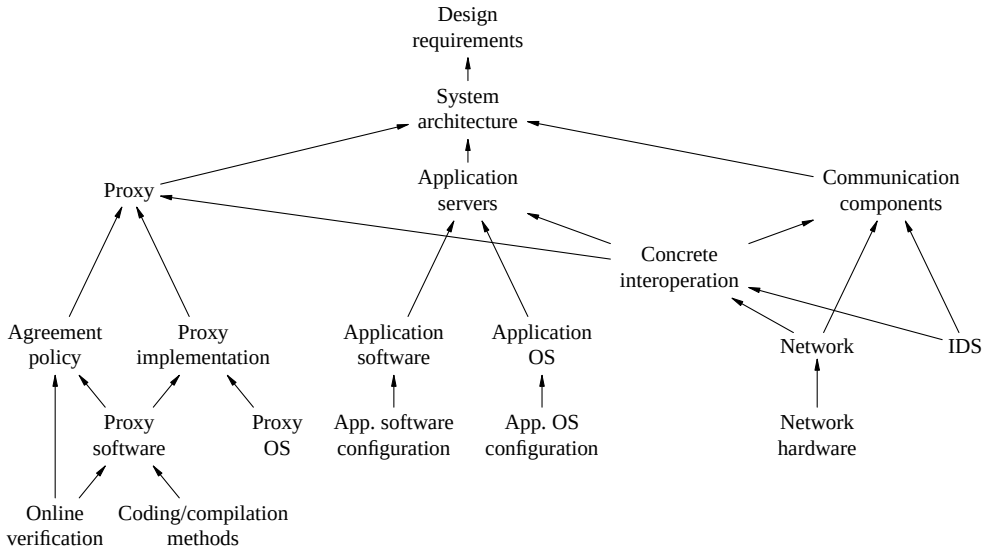

Figure 3: Structure of the design assurance argument for the DIT system.

4.2 An IAC for an Intrusion-Tolerant JBI

In the OASIS Dem/Val program sponsored by DARPA IPTO, an architecture for a highly survivable exemplar Joint Battlespace Infosphere (JBI) has been designed and is now being implemented as a prototype system for demonstration and evaluation. This particular intrusion-tolerant JBI (IT-JBI) is currently referred to by the name DPASA,[9] after the BBN-led project that designed it.

The DPASA design combines state-of-the-art COTS technologies, such as managed switches, modern databases and programming platforms; DARPA-developed survivability technologies, such as distributed firewalls, autonomic response mechanisms, distributed middleware, intrusion detection and alert correlation, Byzantine tolerant protocols, and cryptographic techniques; and design principles, such as containment of attack effects, isolation of compromised parts of the system, and the application of redundancy, diversity, and dy-

[9]DPASA stands for Designing Protection and Adaptation into a Survivability Architecture.

namic adaptivity. Individually, each of these is an incremental improvement. Working together and augmenting one another, they represent a significant and serious effort to advance the state of the art in the development of survivable systems.

A JBI consists of clients interacting via mechanisms of publish, subscribe, and query (PSQ), with this interaction mediated and supported by a JBI platform providing middleware services. A JBI client is a mission application that is enabled to use the JBI platform through the CAPI (Common Application Programming Interface) for its PSQ interaction with other clients. In a typical implementation, a portion of the platform is integrated with the JBI clients while the rest of the platform, the core, exists on its own, independent of any client, and implements the publish, subscribe, and query operations as services to the clients. It is possible to view the survivability architecture for a JBI as a specific instance of the more general framework for designing survivable distributed systems.

In the DPASA context, the JBI core is viewed as being controlled more extensively than the JBI clients, and the design reflects that in the form of a highly available and well-protected core. A distributed middleware layer manages the interaction between the clients and the core. The clients, which are more numerous, also employ hardening, redundancy, and adaptation measures, but they are tightly monitored by the more trusted core. The whole system is instrumented with intrusion detection sensors, with a sophisticated correlation mechanism as part of the core.

4.2.1 A Defense-Enabled JBI

A detailed description of the DPASA IT-JBI is beyond the scope of this report, since DPASA is not directly connected to the Cyberscience research, except as it serves as a test case for IAC development. Nevertheless, some familiarity with the DPASA design is necessary to understand the design-level IAC we have developed. Here we present a brief

overview of the DPASA design. For more details, the reader is referred to the DPASA Phase I Design and Validation reports [Pal03, San03].

One of the main JBI objectives is to facilitate easier, quicker, and on-demand integration of disparate applications in support of a mission. The JBI aims to achieve this objective by treating a mission operation as loosely coupled interactions between information producers and information consumers. This publish-subscribe interaction between the information producers and subscribers, collectively called the JBI clients, is further augmented with a query capability where consumers can query for information pertaining to some topic or request a specific information product. The event channel abstraction underneath the publish-subscribe paradigm, as well as the information object (IO) repository required to support the query, implies a logical hub-and-spoke architecture for the JBI. The JBI vision further strengthens this logical hub-and-spoke view by including the notion of a JBI platform that hosts the services to be used by the JBI clients and also services for managing a JBI. Figure 4 depicts the high-level structure of a notional JBI.

The DPASA view of the JBI platform consists of thin client-resident parts with the bulk of the platform services implemented in a core. In addition to the existing platform services in the baseline (the JBI exemplar developed by AFRL), the defense-enabled JBI has services that lie in the core, such as management of the various survivability mechanisms introduced by the survivability architecture. These services are organized as layered zones, with the idea that most critical services are protected by multiple perimeter boundaries, as depicted in Figure 5.

Zones create concentric barriers between clients and critical core services. The design objective is to force an attacker to compromise a host in each zone, without being detected, in order to mount an attack across zones. To prevent flow-through attacks that could leap across zones, client-to-core transactions that cross zones are proxied at the communications level, the middleware level, and in most cases at the applications level. Thus, an attacker

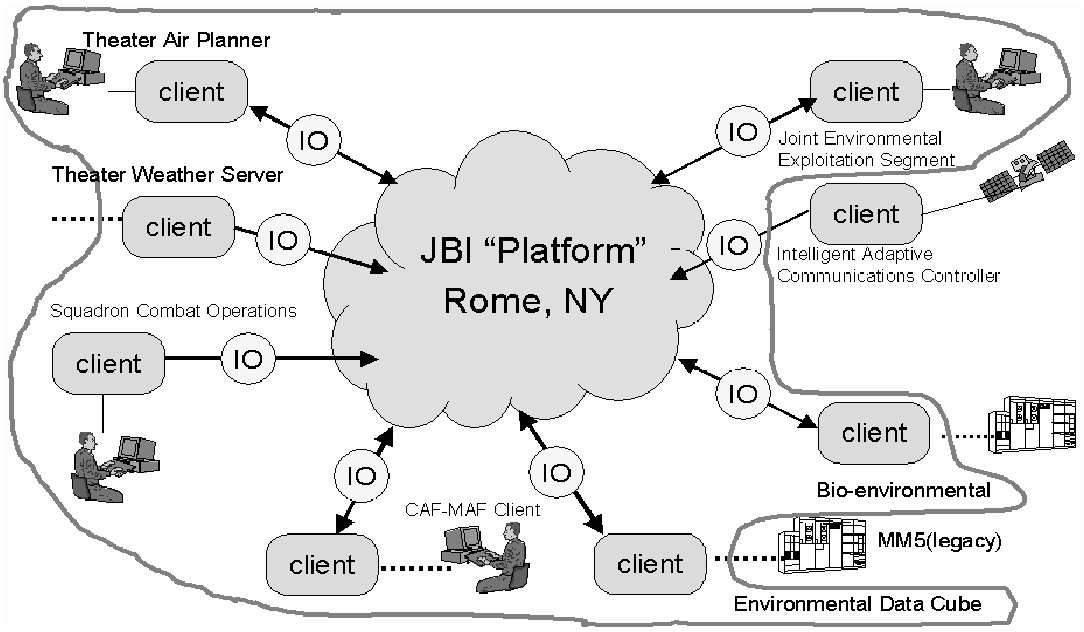

Figure 4: Structure of a notional JBI.

who succeeds in compromising a client would then have to compromise one of the core access proxies (in the crumple zone), and then compromise a host in the operations zone, in order to mount an attack on a system manager, all while remaining undetected.

A network protection domain, provided by enhanced, encryption-enabled network interface cards (called ADF NICs) whose behavior is governed by special network policy servers, severely limits the ability of an attacker to mount integrity or confidentiality attacks against the system from the wide-area network (WAN). Plausible attacks from outside the system would have to originate over back-end connections to clients, thus forcing the attacker through multiple zones in order to compromise the innermost core. Figure 5 shows the details of the physical architecture within what is called a core *quadrant* or *channel*.

The zones are populated as follows:

- The client zone contains all the various client hosts. These hosts cannot communicate directly with each other in normal operation.
- The crumple zone contains the core access proxies. These proxies constitute the first line of defense between the core and clients.
- The operations zone provides the JBI publish, subscribe, and query functionality as well as various information assurance functions, including the policy servers to manage the ADF NICs, alert correlation, and intrusion/fault detection mechanisms, such as the Guardians and the network intrusion detection system (NIDS).
- The executive zone contains the system manager that has the job of coordinating the operations of the other JBI components, managing the overall status of JBI clients, and providing the primary interface for the JBI managers (the JBI CIO's staff).

The hosts in a core channel communicate with each other over a managed switch. The switch has a hardware port-blocking capability that controls the ability of hosts to communicate directly with one another. This provides for enforcement of the zone structure (note, in particular, that the access proxy cannot directly talk to the system manager) and limits the effects of flooding attempts within, and among, core quadrants.

The core is composed of redundant channels that provide intrusion/failure detection capability and reserve resources that are used if the capabilities of one or more of the channels are compromised. The level of redundancy is a trade-off between cost/manageability and robustness of operation/intrusion detection. The DPASA design utilizes four core channels (quadrants) to provide Byzantine tolerance for intrusions that may reach the executive zone.

Clients communicate over a network with the DPASA core via one or more of the core access proxies, depending on the particular protocol being used. In the event that multiple compromises render quadrants temporarily inoperable or inaccessible, the protocols

will adapt to the number of available quadrants. The redundant core quadrants employ diversity across the crumple zone to make it unlikely that they would all be compromised simultaneously by exploitation of a single vulnerability.

Cross-quadrant communications are strictly limited and exist to support three specific capabilities: 1) Robust PSQ protocols that guarantee that IOs will be properly handled by the core even if multiple quadrants become inoperative due to failure or compromise; 2) Byzantine-tolerant interactions among the system managers that guarantee that overall core behavior will be properly managed/coordinated if one system manager is compromised, and that good (although not Byzantine-resilient) core management/coordination will still be available if multiple system managers are down; and 3) coordination among policy servers.

4.2.2 The Design-Level IAC for DPASA

The DPASA IT-JBI has provided an important test of the security co-design and IAC concepts, resulting in significant refinements of both concepts. We now present an overview of the design-level IAC for the DPASA IT-JBI. The development of this IAC was supported largely by the DPASA project under the sponsorship of the OASIS Dem/Val program, and was performed by a large team from several organizations. The representation of the IAC in SEAS was performed as part of the Cyberscience research.

The IAC for the DPASA IT-JBI is large and complex. The full, textual version of the IAC was produced by the DPASA project and is presented in that project's final validation report [San03]. The SEAS representation of the IAC is also large and complex and cannot be completely and legibly presented in this report. The SEAS version of the IAC is best viewed interactively in SEAS itself. The SEAS objects that capture the DPASA IAC, along with a version of SEAS with installation and use instructions, are provided as a separate

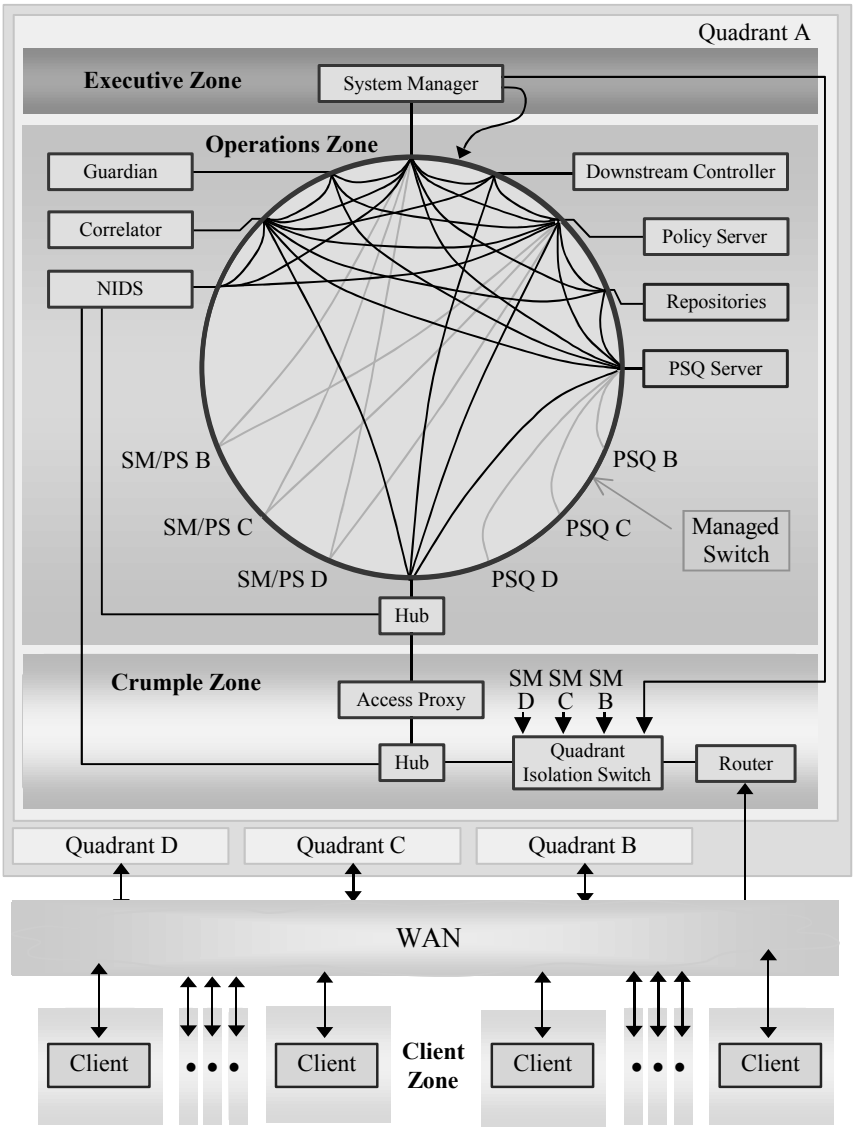

Figure 5: Architecture of a DPASA IT-JBI core quadrant.

deliverable.[10]

Types of Evidence One of the major differences between the IACs for DPASA and DIT is in the diversity and volume of evidence assembled, which is much greater in the DPASA case. This is partly the result of the scope and scale of DPASA itself, which is a more complex system. It is also partly the result of more stringent and specific survivability requirements for DPASA, many of which are expressed quantitatively. The presence of quantitative requirements poses a significant challenge in the development of the IAC, especially at the design stage, when there are few, if any, artifacts (such as hardware and software) that can be directly evaluated and measured. In the case of DPASA, many of the quantitative aspects of survivability were evaluated through simulation modeling, using an abstract mathematical model of the system that aimed to be faithful to the design throughout the evolution of the system.

Other aspects of the design could be measured more directly, since the design specified the use of certain well-developed and well-understood components, such as intrusion detection systems and policy-driven network interface cards. Still other aspects of the design could be dealt with in more abstract terms, using structured arguments (at various levels of formality, ranging from plain English to mathematical logic) and exploratory approaches, such as whiteboarding.

These various forms of design validation were applied according to their appropriateness,

[10] As noted earlier, SEAS was developed partly under DARPA sponsorship by SRI's Artificial Intelligence Center (AIC). SEAS was originally developed to operate on Sun hardware running the Solaris operating system. To facilitate the use of SEAS in open-source computing environments (including SDL), the Cyberscience project produced a version of SEAS patched to operate on PC hardware running various versions of the GNU/Linux operating system. This patched version of SEAS is provided to the Government under the same terms as the original version.

given the specific characteristics of the design elements being evaluated and the nature of the requirements placed on those design elements. More specifically, the following kinds of evidence were integrated into the IAC:

- *Logical argumentation.* Construction of a plain-English structured argument alleging that a stated property of the design is true. In particular, if the property is of the form the design satisfies requirement R and R is not quantitative, this technique can be used to establish validity. Logical argumentation is also useful in verifying a logical decomposition of a requirement into subrequirements (provided the requirement admits such a decomposition).

- *Probabilistic modeling.* Construction of a simulation model of the system (based on design documentation) as it operates in a representative use/attack environment. Solution of the model (values of measures defined on the model) can then determine whether a given requirement (with nontrivial probabilistic quantification) is satisfied.

- *Experimentation.* Experiments with actual system components or prototypes. Results obtained here can sometimes be elevated so as to validate the design with respect to a higher-level requirement. Experimental results are also used to estimate parameter values used in simulation models.

- *Threat and vulnerability assessment.* Analysis of effects due to possible threats to the system or vulnerabilities in the system. Results obtained can increase confidence in the design (if positive) or suggest design modifications (if negative). Results of such assessments are also useful with regard to representing attack effects in probabilistic models.

- *Whiteboarding.* An approach for evaluating the relative strengths and weaknesses of a system design or implementation. In the context of our validation effort, it can be

viewed as an experiment that tests the following hypothesis: The IT-JBI design meets its assurance requirements. Accordingly, this activity is intended to gather data that convincingly supports or refutes this hypothesis.

The evidence referred to at the outset is design related in the sense that it testifies to desired properties of the design itself (or existing component implementations) that the practices followed during the design's conception. Accordingly, it does not refer to peripheral evidence, such as the reputation or experience of the designers, although such evidence could certainly contribute to a convincing IAC.

Design-related evidence can be further classified according to whether or not it testifies to satisfying a specific requirement. Such a requirement can be at the top level (mission- or system-level requirements) or at a lower level that leads to satisfaction of a higher-level requirement. More precisely, evidence is requirement related if, for some specified or derived requirement concerning the design in question, it attests to the assertion that the design satisfies the requirement (or, indeed, if it attests to the contrary).

Other types of design-related evidence need not be linked to a specific requirement, for example, evidence that testifies to the design's general ability to prevent or tolerate intrusions due to various types of attacks. Results of threat/vulnerability analyses and whiteboarding are typically in this category.

DPASA IAC Development Generally, a requirement at a relatively high level is first decomposed into subrequirements by iterating a process of logical decomposition resulting in a decomposition tree for the root requirement, as described in Section 2. In turn, such a tree can usually be simplified by coalescing multiple occurrences of a subrequirement, in which case the tree is technically a directed acyclic graph (DAG).

An essential ingredient of any validation effort is carefully stated design requirements that

accurately and unambiguously capture desired properties of an eventual implementation. The principal requirement for the DPASA IT-JBI design is the following:

R1. The design, when implemented, will provide satisfactory mission support under real use scenarios and in the face of cyber attacks.

Although this requirement is rather obvious, it requires a more precise definition of terms such as "satisfactory mission support," "use scenarios," and "cyber attacks" to obtain convincing evidence that R1 is indeed satisfied. To that end, several high-level requirements of a more specific nature are the following survivability goals of the JBI exemplar, as stated in the OASIS Dem/Val Proposer Information Pamphlet (PIP), and which are referred to as the "PIP requirements":

PIP-1. Provide 100% of JBI critical functionality when under sustained attack by a Class-A red team with 3 months of planning.

PIP-2. Detect 95% of large-scale attacks within 10 minutes of attack initiation, and 99% of attacks within 4 hours, with less than 1% false alarm rate.

PIP-3. Prevent 95% of attacks from achieving attacker objectives for 12 hours.

PIP-4. Reduce low-level alerts by a factor of 1,000 and display meaningful attack state alarms.

PIP-5. Show survivability versus cost/performance trade-offs.

Note that PIP-5 is irrelevant with respect to R1; that is, mission objectives can be satisfied whether or not the tradeoff goal is accomplished.

The PIP requirements are then grouped and decomposed as depicted in Figure 6. Although the remainder of the IAC structure cannot be depicted legibly here in its entirety, a sense

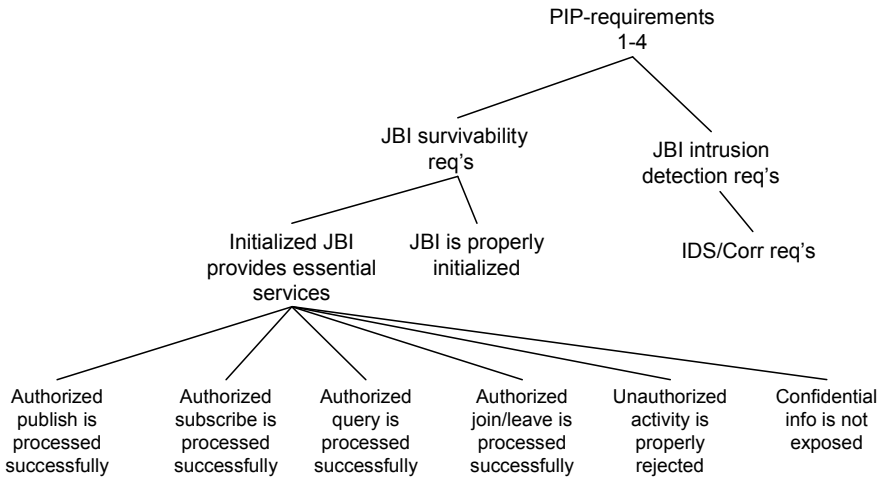

Figure 6: Decomposition of top-level requirements.

of its size, scope, and general structure can be gleaned from the graph in Figure 7. An interactive, browsable version of the IAC structure is supplied with the SEAS software provided as a separate deliverable.

The DPASA IAC is currently not complete enough to make firm conclusions about the whether the DPASA design, when implemented, will meet the JBI survivability objectives. Not all areas of the IAC structure have been completely fleshed out, even at the design level. In those parts of the IAC that have been detailed as far as the design permits, evidence remains subject to human interpretation, and in the quantitative assessments, significant uncertainty remains. In the SEAS representation of the IAC, this uncertainty is directly reflected in the assignment of values representing strength of evidence. That is, on the SEAS scale of 1/green (which, for IACs, we interpret as meaning *very likely*, in response to a positively phrased question pertaining to survivability) to 5/red (interpreted as *very unlikely*), we represent uncertainty as 3/yellow (meaning *as likely as not*). One might argue that a conservative estimation of evidential strength calls for the assignment of a value such as 5/red in the absence of strong evidence to support an IA claim. However, we have taken the

view that, in the absence of evidence, a claim is neither supported nor refuted. The values 5/red and 4/orange are reserved for cases in which the available evidence argues *against* an IA claim, while the values 2/yellow-green and 1/green are used when the evidence supports the claim. To ensure that uncertainty does not lead to a false sense of support for high-level IA claims, we use strength propagation rules (see Section 3.2) that appropriately account for the uncertainty at the top level.[11]

[11] In fact, SEAS is intended to capture uncertainty directly, through the assignment of multiple values that represent the range of possible responses (or through the assignment of no value at all, meaning complete uncertainty). However, we have found that the available fusion methods in SEAS do not propagate such uncertainty in a way that we find intuitive for IACs. We have therefore opted to "overload" the middle value (3/yellow) to represent uncertainty, which we concede is itself not an optimal solution.

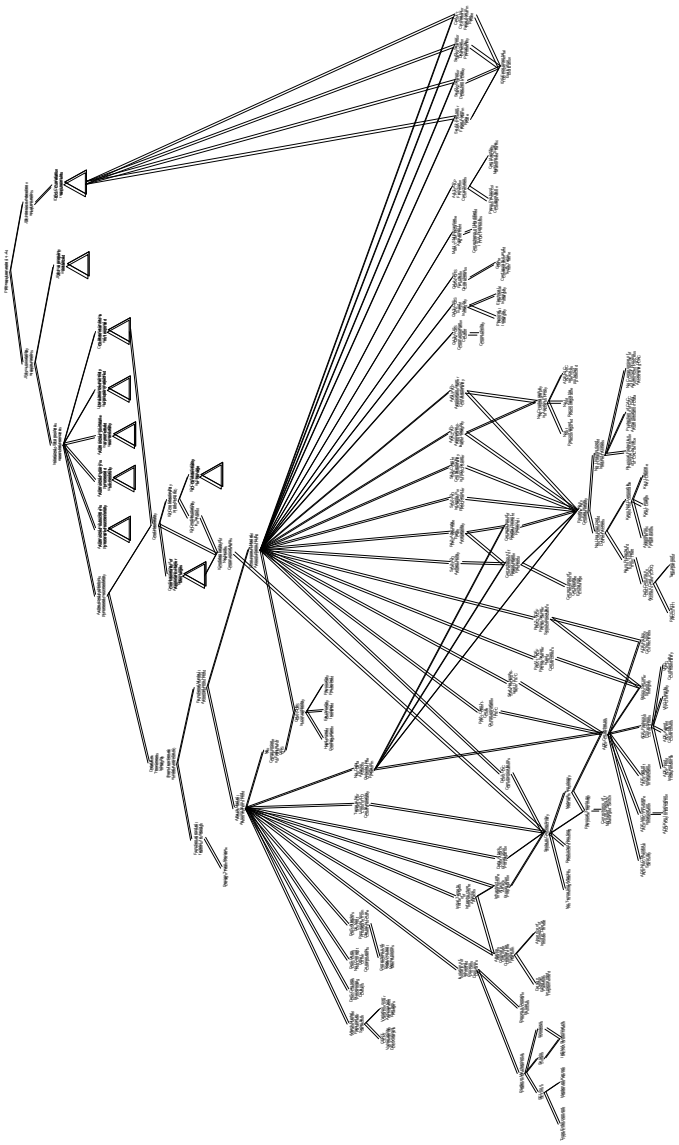

Figure 7: High-level structure of IAC.

5 Alternative Approaches

Many research efforts have been conducted in order to overcome the lack of a global and commonly agreed upon process to evaluate whether a system satisfies its security requirement. Such a process would include collecting and structuring evidence supporting the claim that the system indeed satisfies its security requirement. Although several partial methodologies have been developed, there is still a lack of a complete methodology that supports the assembly of a comprehensive IAC. In practice, however, many more or less advanced methodologies have been proposed and used as a means for certifying computer systems used in critical areas such as avionics and defense.

Our own research is related to several research areas:

- Defining guidelines for building systems that should be secure
- Defining and capturing system security requirements
- Discovering methods for evaluation of the security of existing systems
- Providing technologies for assembling evidence

5.1 Engineering Guidelines for Secure Systems

Research in defining guidelines for building secure systems and providing evaluation standards produced an effort of standardization called the “Common Criteria” [Com98].

The Common Criteria project harmonizes European, Canadian, and U.S. Federal Criteria into the Common Criteria for Information Technology Security Evaluation for use in evaluating products and systems and for stating security requirements in a standardized way. Increasingly, it is replacing national and regional criteria with worldwide criteria accepted

by the International Standards Organization. The U.S. Department of Defense published the first criteria in 1983 as The Trusted Computer Security Evaluation Criteria (TCSEC), more popularly known as the "Orange Book" [dod85]. The current issue is dated 1985. U.S. Federal Criteria were drafted in the early 1990s as a possible replacement but were never formally adopted.

The Common Criteria project identifies seven levels of evaluation assurance, and aims to develop evaluation criteria for all seven layers that can be applied to any security-critical system. While the Common Criteria approach focuses on defining standards for expressing security requirements and evaluations of systems against those requirements, our focus has been on the complementary process of assembling the evidence and supporting arguments into an IAC. Our work is not tied to a particular standard, and can include a variety of arguments derived from different evaluation methodologies. Our aim is to go beyond the standardization effort by allowing the assembly of evidence collected by different methodologies in a comprehensive and useful IAC structure.

5.2 Security Evaluation

Several technical advances have been reported by researchers in providing tools and methods for checking whether a software system satisfies its security requirement, and therefore provide evidence and evaluation techniques. Some of those evaluation techniques are model based. A model of a part of a system is built and checked for nonconformance to a particular security requirement. Model-based techniques have been used to check security protocols [DY83], enforce security policies [Sch00] in operating systems, and model various aspects of system dependability [DDD+00]. Code-based evaluation techniques use the concrete implementation in place of a more abstract model and check the source code for security vulnerabilities. Tools such as Cyclone [JMG+02], CCUred [NMW02], and

StackGuard [CPM+98] use dynamic typechecking to check source code for buffer overflow vulnerabilities, and insert run-time checks. Fault injection techniques [GOM98] are a more general way of checking a larger class of security vulnerabilities including buffer overflow.

5.3 Security Requirements Engineering

While there is general agreement that security requirements engineering is difficult and needs more attention, there is much less agreement on how it should be done, as demonstrated by a recent symposium devoted to security requirements [Pur01]. Although there are compelling arguments for significant use of formal methods in the development of security requirements [Rus01], current approaches (at least those described in the open literature) tend to be informal (mathematically speaking) and social, guided by developmental methodologies, and informed by general security principles derived from experience, standards, and common security practices [Irv01, Ste01, AVP01, SW01, Gas01]. Often there is a tendency for security requirements development (and the practice of security in general) to focus more on mechanisms needed to address specific threats and vulnerabilities, rather than on the overarching objectives of security requirements in terms of their role in meeting mission objectives. As a result, the security requirements as stated in a specification document may lack context and traceability, both of which are critical, not only to provide confidence in the correctness of the security requirements themselves, but also to enable more rapid isolation and resolution of failures to satisfy them. Our proposed co-design approach explicitly accounts for context and requirements traceability through the use of the co-design object base as its central component.

5.4 Technologies for Assembling Evidence

5.4.1 Safety Case Development

In the safety domain — as in security — it is usually impossible to show with certainty that a system is absolutely safe. Instead, one must demonstrate that the system is sufficiently safe for its purpose. Such a demonstration cannot be a pure "mathematical proof of safety" (although mathematical proofs can be part of it), but a convincing explanation, supported by evidence, of why the system is safe enough. A *safety case* is a document where such a demonstration is developed. More precisely, a safety case can be defined as

> *a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application, in a given environment [BB98].*

It helps to think of a safety case as a document intended to *convince* a rational but skeptical person with adequate expertise (say, from a regulatory organization) that a system is sufficiently safe for being deployed and used.

A safety case is then about a particular system in a particular environment. It makes safety-related claims about the system, produces supporting evidence, and develops an argument that the evidence indeed supports the claims. The assumptions and judgments underlying the argument should also be clear and explicit. Examples of relevant pieces of evidence may be the design process or tools used, results from fault-tree analysis or other safety analysis techniques (cf. [Lev95]), quantitative data about the failure rates of hardware components or the amount of testing done (testing coverage), proof of correctness of part of the design, and past field experience [BB98, WKM97]. A difficulty is to combine heterogeneous pieces of evidence to form a coherent and convincing argument that the system is adequately safe.

Specialized tools have been developed to construct and maintain safety cases [BBE+98, FCM93], but they are essentially structured editors. They help organize and present a safety case, typically in a diagrammatic fashion, but do not help ensure the soundness of the argument. Generalist tools for developing structured arguments [LHR01] may be more effective in producing convincing safety cases.

Safety cases provide a model for how heterogeneous data can be used as evidence in a non-demonstrative, but convincing, argument that a system has a desired quality attribute.

5.4.2 Software/Hardware Co-design

Software/hardware co-design emerged in the 1990s as the integrated design of systems implemented using both hardware and software components [Sub93, GD93, MA93, KAJW93, HDMT94, CGJ+94]. A typical co-design process derives a mixed hardware/software implementation from a single system description. This process involves producing and analyzing specifications, defining and evaluating architectures, partitioning functions between hardware and software components, and implementing the components. Co-Design frameworks [Lee01, LLEL02, BCG+97, LSV98] now include comprehensive sets of verification, simulation, and synthesis tools supporting the design of complex heterogeneous embedded systems. These co-design frameworks have in common the use of a high-level system specification (often combining heterogeneous specification languages) as the starting point to a rational design, and support for examining, evaluating, and verifying alternative architectures, partitions, and implementations [LRS+00, OB98, CB98, Ber91, KAJW93, GSK+01].

Software/hardware co-design provides a model of how a co-design process can address the tradeoffs required in resolving goals in different design dimensions in a separate-but-coordinated fashion.

5.4.3 KAOS

KAOS [DvF93, DDMv97, vDM95] is a goal-driven requirements engineering method developed at the Université Catholique de Louvain. KAOS is aimed at supporting the whole process of requirements elaboration, from the high-level goals to be achieved by a system to the objects, operations, and requirements to be implemented. The KAOS language includes a variety of requirement elaboration concepts, such as goals, requirements, assumptions, agents, views, operations, and scenarios.

KAOS supports a systematic requirements elaboration method based on the identification and refinement of system goals, the identification of operations and agents relevant to the goals, and the assignment of operations to agents. Agents are the active entities of a system. Each agent is responsible for achieving some goals by performing appropriate operations. Agents can be implemented by software and hardware, but a system may also include human agents or mechanical devices. Environmental assumptions may be discovered along the way: typically, responsibilities that are assigned to users or other agents outside of the software system become environmental assumptions. Goal refinement plays an essential role. Refinement links are structured in an AND/OR relationship that allows one to explore and document different refinement alternatives.

The requirements elaboration process is guided by meta-level knowledge explicitly captured in KAOS. The meta-level provides a rich taxonomy of goals, objects, and operations with associated heuristic rules and constraints. Heuristics indicate how a certain type of goal may be refined. For example, the meta-level introduces the category of *safety goals*, which are a subclass of *avoidance goals* and can be refined into *hard requirements* [vDL98].

KAOS provides a useful conceptual starting point for the most abstract stages of security co-design, where system requirements are derived from mission goals. Its main shortcoming is that the concepts it provides are specifically oriented toward functionality rather than

security. As a result, security-specific concepts — vulnerabilities, for example — are not easily represented, and the notion of requirements elaboration is basically the classic notion of refinement, which guarantees presentation of allowed behavior, but not preservation of excluded behavior.

5.4.4 UML

The Unified Modeling Language (UML)[BJR99, FS00] is the most widely used specification notation in industry. It combines concepts from a number of previously popular methods — including, most prominently, Grady Booch's work, Ivar Jacobson's Object-Oriented Software Engineering (OOSE), James Rumbaugh's Object Modeling Technique (OMT), and David Harel's Statecharts — in a unified framework. Initial work on UML began in 1994 at Rational, and a standardization effort was initiated by the Object Management Group (OMG) in 1995. The OMG standard for UML version 1.0 was published in 1997. Among the contributors to the standard were the Digital Equipment Corporation, Hewlett Packard, IBM, Microsoft, Oracle, and Texas Instruments. The diversity of notations in UML facilitates hierarchical specification. For example, the structure of the system architecture can be specified using class diagrams, and further details of the interaction, such as communication protocols, can be specified using statecharts. In addition, many of the notations — including both class diagrams and statecharts — support a notion of horizontal refinement, where a single "box" (i.e., class or state) can be expanded into a network of "boxes and arrows" (i.e., a class diagram or statechart).

Because UML is a *de facto* standard, it provides a good model of how diverse functional models should be integrated in the COB.

5.4.5 Ada Programming Support Environments

As the design of the Ada programming language matured in the late 1970s, it became clear that some of its features — such as separate compilation and the `INCLUDE` and `USE` constructs — entailed that compilation must be supported by a somewhat more sophisticated environment than those of the languages it was intended to replace (primarily, Fortran and assembler). Thus, an effort to specify the features that an Ada Programming Support Environment (APSE) should include was initiated. Three months prior to the official release of the final version of the official APSE specification [(HO80], called "STONEMAN", in February 1980, a workshop was held in San Diego, California, to provide a forum for a large number of representatives of industry, academia, and the government to comment on a preliminary version of the document. The principal conclusion of the workshop was that there was no consensus as to what an APSE should look like, and that STONEMAN should not be taken to be a set of firm requirements, but rather an ideal toward which APSEs should aim.

Clearly, creating a full-featured APSE with powerful system engineering support tools would require tremendous effort. Although the STONEMAN specification called for every life cycle artifact, from early informal statements of requirements through the current configuration, to be stored in the APSE database, the Ada market never grew sufficiently to support development of this capability by vendors. Although the value of retaining this information was generally agreed upon, and sophisticated tools that made use of the information were hoped for, the utility of the record was never established in practice.

From the perspective of the proposed research, an APSE database would be deficient in two respects. First, the emphasis is placed on storing the results of applying a software development process (in our case, security co-design), rather than storing a model of the process itself and a record of how it was applied in the particular case. Thus, crucial information

about the source of information (i.e., where it came from and why) and interrelationships among various bits of information would be lost. Second, no emphasis is placed on the necessity of separating information regarding functionality from information regarding security and other dependability properties. There is also a general problem with the APSE perspective: it assumes that the software for a system will be developed within a single environment. If it is assumed that the present practice of different developers using different environments — different editors, different compilers, and even different programming languages — continues, the COB must be more loosely coupled to the environment.

5.4.6 Knowledge-Based Software Assistant

The report on a knowledge-based software assistant (KBSA) [GLB+83] outlined an ambitious approach to developing an expert system to mediate and support all life cycle activities. In particular, all decisions concerning requirements, design, validation, implementation, testing, and maintenance were to be recorded in a computerized "corporate memory". The rationale for the decision was to be included as well. Subsequently, the Rome Air Development Center (now, the Air Force Research Laboratory at Rome) funded development of prototype KBSA "facets" for requirements analysis, specification, development, performance evaluation, testing, and project management. While the goals of the original report were not realized, the prototypes provided a technology base for subsequent development of very successful, but more limited, tools, such as Amphion at NASA Ames [BFH+99] and the Kestrel Institute's Planware plan synthesis system [BGL+98]. These two tools have primarily focused on domain-specific software synthesis, that is, on building a domain-specific implementation "facet". Other researchers have focused on building domain-specific versions of other facets. However, there seems to have been little research and development effort devoted to creating a domain-specific version of the KBSA infrastructure. Our proposed development of a corporate memory tailored to the

needs of security co-design and an explicit knowledge-based representation of the security co-design process, designed for use with conventional development tools rather than KBSA facets, represents “simplification for achievability” of the KBSA vision in a new dimension.

5.4.7 Literate Programming

The literate programming movement, inaugurated by Donald Knuth [Knu94], represents an attempt to capture the rationale for a program’s design and its source code in a single document. Both nicely formatted human-readable program documentation and the source code that is provided to the compiler are generated from this single document. The primary weakness of this scheme, from the point of view of the proposed effort, are

1. the language in which the document is written is designed specifically for the purpose of generation of documentation and source code, rather than aiming at a more general-purpose representation of information from which other artifacts — a requirements specification, a test suite, and so on — could be generated,

2. the data structure used, a text file, is not suited to storage and retrieval as a collection of objects in a database,

3. the recorded rationale is in the form of natural language text, and is thus ill-suited to play any useful role in analysis tools,

4. there is no tool support to encourage recording the complete rationale or to assist in appropriately structuring it, and

5. what is recorded is typically a final, polished version of the rationale, rather than a complete historical record of the search for that rationale.

While the technology base provided by literate programming is inappropriate for our effort, it is worth noting that the basic idea of recording the rationale for a program as it is developed has led to software of impressive quality. Knuth's TEX typesetting system, for example, is generally believed to contain fewer errors than any other program of comparable size. It seems reasonable to expect that simply recording the security aspects of the system design process in a form suitable for external review will lead to similar improvements in security levels, even prior to the development of analysis tools.

6 Lessons Learned

The Cyberscience project set ambitious goals for advancing the science and practice of secure systems development. Although many of our longer-term goals remain to be met, we believe that our security co-design methodology and accompanying work on the development of information assurance cases represent advances in the field of information assurance, both in terms of our understanding of the challenges in the development of high-assurance systems and in terms of practical methods for developing such systems.

Of course, research results are rarely unambiguously positive, and ambitious goals are usually not met completely. In this sense, the results of Cyberscience are not exceptional: there are positive, affirming results, and other results that point to a clear need for further research.

6.1 Security Co-design

On the positive side, we believe that our primary hypothesis for developing the co-design methodology — namely, that security must be an integral part of secure systems development — has been supported fully by our experiences in applying it to three different systems.

Our first case study in security co-design and IAC development was Genoa CrisisNet, which was at the time the central storage management component of the Genoa crisis prediction and analysis system.[12] We realized at the time that using Genoa CrisisNet as a case study could be problematic, since it was already in an advanced state of development, and thus, could not easily serve as an example of security co-design, since the developers had clearly not applied co-design principles, and we were not at liberty to undertake the

[12] Somewhat coincidentally, SEAS is a component of Genoa.

design from scratch. However, in attempting to construct an assurance argument for CrisisNet, we decided to assume a "clean slate", developing an abstract design from top-level requirements (which, unfortunately, had to be reverse-engineered from Genoa use cases and implemented components). This process quickly uncovered a fundamental weakness in the design of CrisisNet that could allow its access control mechanisms to be subverted, providing a clear example of the results that can be expected from failure to account for security requirements at the outset.[13]

In the DIT and DPASA cases, we were fortunate to have been involved early enough in the design process that co-design principles could be applied in an integral way. Although a full IAC for DIT was never undertaken, the system was demonstrated to meet high-level, nonquantitative assurance goals, through its ability to serve correct Web content in the face of successful cyber attack on some components (intrusion tolerance). For DPASA, the jury is still out; the system is under development and is expected to be subjected to evaluation by concerted red-team activity in early 2005. Nevertheless, the design-level IAC did provide convincing evidence to support claims that a system built to DPASA design specifications would, with high probability, satisfy its survivability requirements. Anecdotal evidence gathered throughout the design phase suggests that feedback from validation (IAC) activity back to the designers did produce observable and measurable improvement in the design with respect to survivability goals.

6.2 Experience with SEAS for IACs

Our use of SEAS in the development of a design-level IAC for the DPASA IT-JBI has identified several strengths of SEAS that we believe should be preserved in any future IAC

[13]Subsequently, and mostly coincidentally, the Genoa project abandoned CrisisNet in favor of a COTS product believed to have better security and survivability characteristics.

development tool, and several significant shortcomings that future tools should strive to overcome.

6.2.1 Strengths of SEAS

Organization and Corporate Memory From the point of view of IAC construction, SEAS's greatest value is perhaps as an organizational tool. The tree-structured arguments that SEAS supports correspond closely to the (acyclic) graph-structured arguments around which our notion of IACs is built. In fact, it is possible, although awkward, to construct our more general graph-based (DAG) argument structures using SEAS. In addition, SEAS supports argument versioning, so that the history and evolution of arguments can be tracked.

Ease of Use SEAS presents a simple and intuitive interface, along with context-sensitive help, which makes it relatively easy to begin using the SEAS tool fairly quickly. At the same time, SEAS can be viewed not only as a tool, but also as a paradigm for argument construction in the intelligence domain. Becoming proficient in the SEAS paradigm requires more time and effort, although significant help and examples are available to assist in the process. However, not all aspects of the SEAS paradigm appear well suited to IAC construction, as discussed in Section 6.2.2.

Support for Controlled Collaboration A useful feature of SEAS, both for its intended application and for IAC construction, is its support for multiuser and collaborative argument construction. SEAS supports user accounts and hierarchical user groups, and enforces access control on SEAS objects (templates, arguments, etc.) based on administrator- and owner-specified access control lists (ACLs) that define who may read, modify, and delete specific SEAS objects. Most important, the SEAS paradigm encourages review and assessment of arguments by multiple users, and the SEAS tool can record these multiple

assessments and highlight areas of agreement, which tends to reinforce confidence in an argument, and disagreement, which may indicate a need to revisit and revise an argument.

6.2.2 Shortcomings of SEAS

Assessment of Soundness and Completeness Perhaps the most significant shortcoming of SEAS, from the standpoint of IAC support, is the absence of any objective means (and mechanism for incorporating such means) for assessing the validity of constructed arguments. In the SEAS paradigm this assessment is left entirely to the judgment of human evaluators. While the use of SEAS's confidence value ranges and simple confidence propagation functions can assist assessors in identifying areas of potential weakness in arguments, these values are rooted in subjective assignments of values and so must yield subjective results. Of course, it should be noted that this approach may be well suited to SEAS's intended application, but for IAC construction, we need something stronger to assist argument developers and assessors in determining (to the extent it is possible to determine objectively) whether assurance arguments are sound and complete.

As an illustration of the issue, consider that SEAS does not make explicit (nor does it allow the user to make explicit) the intended meaning of the branching of a node in the argument graph. For example, take the common case where one intends the branching of a claim A into subclaims $A_1, \ldots, A_n$ to be a conjunctive implication; that is, claim A holds if each of subclaims A_1 through A_n holds. As a start, one might desire that the tool simply evaluate the implication; that is, determine a truth value for A based on the truth values of A_1 through A_n.[14] But to assess completeness of support for the claim, one must establish

[14] In fact, for this case, it is possible to emulate the desired behavior in SEAS. One can select a binary range (*green* / *red*) for the values of A, $A_1, \ldots, A_n$, and the fusion method *maximum* (where *red* is "stronger" than *green*). The value automatically assigned to A will be *green* (true) if each of A_1 through A_n is assigned *green*; otherwise, A will take the value *red* (false).

that the *implication* is true; that is, we must show that the truth of all subclaims $A_1, \ldots, A_n$ is sufficient to establish the truth of claim A. To provide automated support for this sort of assessment, we require logical specification and reasoning capabilities ("theorem provers", if you will) for experts to use in formulating their arguments. A significant challenge will be to make such capabilities available without severely compromising the tool's ease of use. Of course, it is unreasonable to expect that novices in formal methods will be able to make effective use of such capabilities as rapidly as they may learn to use SEAS. However, we believe that a reasonable compromise would be to make a "library" of the most IAC-relevant automated reasoning methods available, along with intuitive interfaces for specifying the intended relationship of nodes in an argument.

We note, however, that more general notions of completeness in the context of IA arguments are impossible to address with absolute certainty, automated or not. For example, for any useful system it is not possible to make a "complete" argument that the system is free of risk to the objectives for the system. However, it may be more feasible to establish that the system is free of a particular set of vulnerabilities within a specific threat model.

Integration and Propagation of Results A particular challenge to the use of SEAS in the DPASA design validation effort has been the need to integrate the results of different validation techniques and to combine these results into meaningful measures of satisfaction of the system's security objectives. For example, a significant part of the validation effort has involved the use of probabilistic modeling, via stochastic activity networks (SANs) [DDD+00], to estimate the availability and integrity of certain critical JBI functions (e.g., any given attempt to publish an authorized information object in the JBI is expected to succeed with probability 0.97). The different measures computed from the SAN models constitute evidence that supports (or perhaps fails to support) higher-level IA claims, and this evidence is incorporated into the overall IAC structure at the appropriate places. When

we have precise, objective (in particular, numeric) evidence to support a claim, we would like to maintain maximal precision as we proceed "up" the graph in order to determine more objectively the strength of support for the highest-level claims. But in SEAS we are forced to make a manual determination of the strength (in terms of the simple 1–5 range that SEAS supports) of that evidence at the node where the evidence is attached, and the desired precision is lost (except in cases where the support is absolute).

Because the SEAS interface is tightly integrated with its Gister-based argumentation engine, it is difficult to extend the reasoning and evidential strength propagation mechanisms to accommodate additional and, especially, user-defined functions for combining and propagating results. Moreover, it is often the case that some piece of evidence includes a specific (perhaps numeric) result that is independent of the strength of the evidence. Thus, we believe it is useful to separate the notions of evidence (results) and evidential strength so that both may be propagated and used appropriately at higher levels in the argument structure. As is the case for reasoning about soundness and completeness, the challenge here will be to include the flexibility to specify result and strength propagation functions without unduly compromising ease of use.

Process vs. Product SEAS emphasizes the *process* of argumentation, particularly in terms of interaction and collaboration. These features are of course essential to SEAS's role in interagency intelligence analysis and crisis prediction, where one of SEAS's major goals is to facilitate collaborative intelligence assessment. These features are clearly of benefit to IAC development as well. However, we find that there needs to be much more attention given to the *product* side of IAC development. In SEAS, the principal product is an interactively browsable argument, and there is only a very limited capability for exporting (fragments of) arguments for use outside the SEAS environment. For IACs, the ability to generate self-contained representations (such as text documents) of full arguments at

user-selectable levels of detail is essential, so that the IAC can be made readily available for possible noninteractive review by outside assessors and be included as part of other IA assessment products, such as certification reports.

Constraints on Argument Structure SEAS requires a high degree of regularity and even symmetry in argument structure. For example, SEAS version 5 requires both that every nonleaf node in an argument template have the same number of children and that the tree be "full", that is, that every root-to-leaf path have the same length. As with other features of SEAS, this usage constraint was motivated by the nature of SEAS's intended application for assisting intelligence analysts in evaluating evidence supporting different hypotheses. The idea is that, by forcing analysts to frame their arguments in terms of questions whose answers have equal weight, analysts would be less likely to overestimate (or underestimate) the relative support each answer provides to the answer for a higher-level question. Similarly, by forcing analysts to develop each "branch" of an argument to the same depth, analysts should be more likely to give comparable amounts of attention to the development of evidence and argumentation to support their answers to questions in the different branches.

These constraints on argument structure may be appropriate for evaluating evidence on intelligence matters, where the evidence tends to be more subjective and the strength of the evidence necessarily more dependent on human interpretation. For IACs, however, we have found that attempting to force arguments into regular and symmetric structures is more likely to skew the results than to balance them. After all, different organizations and different applications typically have different overall security objectives and therefore place different emphases on different security attributes such as confidentiality, integrity, and availability. These differing emphases, together with the varying availability and strength of evidence to support IA claims, tend naturally to drive the structure of IA arguments to

be less regular and more asymmetric.

It should be noted, of course, that SEAS's strong constraints on argument structure have been relaxed somewhat in version 6, largely in response to feedback from both the intelligence community and our own IAC work. Still, the remaining constraints pose a significant practical impediment to the rapid creation and evolution of free-form argument structures, which are best suited to IACs.

7 Acknowledgments

Key contributors to the Cyberscience project are Victoria Stavridou, Steve Dawson, Bruno Dutertre, Fred Gilham, Joshua Levy, Bob Riemenschneider, Hassen Saïdi, and Tomás Uribe. Special thanks go to John Lowrance of the SRI Artificial Intelligence Center for his invaluable advice and assistance with Genoa CrisisNet and SEAS. We also thank Ian Harrison, Tom Lee, Andres Rodriguez, and Eric Yeh for significant technical assistance with SEAS. Finally, we thank the many members of the DIT and DPASA teams for their collaborative efforts in the formulation of information assurance cases for their respective systems.

References

[AVP01] M. Abrams, J. Veoni, and F. Parraga. Application of the Protection Profile to Define (Security) Requirements for a Telecommunications Service Contract. In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[BB98] P. Bishop and R. Bloomfield. A Methodology for Safety Case Development. In *Safety-Critical Systems Symposium*, pages 194–203, Birmingham, UK, 1998.

[BBE+98] P. Bishop, R. Bloomfield, L. Emmet, C. Jones, and P. Froome. *Adelard Safety Case Development Manual*. Adelard, London, 1998.

[BCG+97] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic Publishers, 1997.

[Ber91] G. Berry. A Hardware Implementation of Pure ESTEREL. Technical report, DEC, Paris Research Laboratory, July 1991.

[BFH+99] W. Buntine, B. Fischer, K. Havelund, M. Lowry, T. Pressburger, S. Roach, P. Robinson, and J. Van Baalen. Transformation Systems at NASA Ames. In *Proc. Workshop on Software Transformation Systems*, May 1999.

[BGL+98] L. Blaine, L. Gilham, J. Liu, D. R. Smith, and S. Westfold. Planware — Domain-Specific Synthesis of High-Performance Schedulers. In *Proc. Thirteenth Automated Software Engineering Conf.* IEEE Computer Society Press, October 1998.

[BJR99] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[BR95] K. Buchenrieder and J. Rozenblit. *Codesign. Computer-Aided Software/Hardware Engineering*, chapter Codesign: An Overview, pages 1–15. IEEE Computer Science Press, 1995.

[BS93] J. Bowen and V. Stavridou. Safety-Critical Systems, Formal Methods and Standards. *Software Engineering Journal*, 8(4):189–209, July 1993.

[CB98] P. Chou and G. Borriello. An Analysis-Based Approach to Composition of Distributed Embedded Systems. In *Proceedings of the International Workshop on Hardware/Software Codesign (CODES'98)*, pages 3–7, Seattle, WA, March 1998.

[CGJ+94] M. Chiodo, P. Giusto, A. Jurecska, H. C. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno. Hardware-Software Codesign of Embedded Systems. *IEEE Micro*, 14(4):26–36, August 1994.

[CMS01] R. Canetti, C. Meadows, and P. Syverson. Environmental Requirements for Authentication Protocols. In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[Com98] Common Criteria Project Sponsoring Organisations. ISO/IEC Standard 15408. *Common Criteria for Information Technology Security Evaluation*, November 1998.
`http://www.commoncriteria.org/`.

[CPM+98] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. Stack-

Guard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proc. 7th USENIX Security Conference*, pages 63–78, January 1998.

[DDD+00] D. Daly, D.D. Deavours, J.M. Doyle, P.G. Webster, and W.H. Sanders. Möbius: An extensible tool for performance and dependability modeling. In B.R. Haverskort, H.C. Bohnenkamp, and C.U. Smith, editors, *Computer Performance Evaluation: Modelling Techniques and Tools: Proceedings of the 11th International Conference (TOOLS 2000)*, volume 1786 of *Lecture Notes in Computer Science*, pages 332–336. Springer, March 2000.

[DDMv97] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, pages 612–623, May 1997.

[De 94] G. De Micheli. Computer-Aided Hardware-Software Codesign. *IEEE Micro*, 14(4):10–16, August 1994.

[DMS95] Kemal A. Delic, Franco Mazzanti, and Lorenzo Strigini. Formalizing a software safety case via belief networks. Technical Report SHIP/T/046, IEI-CNR, Pisa, Italy, August 1995.

[dod85] Trusted computer system evaluation criteria [orange book]. Department of Defense, Standard DoD 5200.28-STD, December 1985.

[DvF93] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20(1–2):3–50, April 1993.

[DY83] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.

[FCM93] J. Forder, Higgins C., and J. McDermid. SAM – A Tool to Support the Construction, Review and Evolution of Safety Arguments. In *Proc. of the Safety Critical Systems Symposium*, February 1993.

[FP91] D. W. Franke and M. K. Purvis. Hardware/software Codesign: A Perspective. In *Proceedings of the* 13^{th} *International Conference on Software Engineering*, pages 344–352, May 1991.

[FS00] M. Fowler and K. Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, second edition, 2000.

[Gar96] D. Garlan. Style-Based Refinement for Software Architecture. In *Joint Proceedings of the Second International Software Architecture Workshop (ISAW-2) and the International Workshop on Multiple Perspectives in Software Developments (Viewpoints'96)*, San Francisco, CA, October 1996.

[Gas01] G. Gaskell. An Analysis of BS7799 and Requirements for eCommerce. In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[GD93] R. Gupta and G. De Micheli. Hardware-software co-synthesis for digital systems. *IEEE Design & Test*, 10(3), September 1993.

[GLB+83] C. Green, D. Luckham, R. Balzer, T. Cheatham, and C. Rich. Report on a Knowledge-Based Software Assistant. Technical Report RADC TR 83-195, Rome Air Development Center, June 1983. Also available as Chapter 23 of *Readings in Artificial Intelligence and Software Engineering*, C. Rich and R. C. Waters (eds.), Morgan Kaufmann, 1986.

[GOM98] A. K. Ghosh, T. O'Connor, and G. McGraw. An automated approach for identifying potential vulnerabilities in software. In *1998 IEEE Symposium*

on Security and Privacy (SSP '98), pages 104–114, Washington – Brussels – Tokyo, May 1998. IEEE.

[GRS99] F. Gilham, R. A. Riemenschneider, and V. Stavridou. Secure Interoperation of Secure Distributed Databases: An Architecture Verification Case Study. In *FM'99 – World Congress on Formal Methods in the Development of Computing Systems, Volume I*, number 1708 in Lecture Notes in Computer Science, pages 701–717, Toulouse, France, September 1999. Springer-Verlag.

[GSK+01] S. Gupta, N. Savoiu, S. Kim, N.D. Dutt, R. Gupta, and A. Nicolau. Speculation Techniques for High-Level Synthesis of Control-Intensive Designs. In *Design Automation Conference*, June 2001.

[HDMT94] X. Hu, J. D'Ambrosio, B. Murray, and D.-L. Tang. Codesign of Architectures for Automotive Powertrain Modules. *IEEE Micro*, 14(4):17–25, August 1994.

[HDRS99] J. Herbert, B. Dutertre, R. Riemenschneider, and V. Stavridou. A Formalization of Software Architecture. In *FM'99 – World Congress on Formal Methods in the Development of Computing Systems, Volume I*, number 1708 in Lecture Notes in Computer Science, pages 116–133, Toulouse, France, September 1999. Springer-Verlag.

[(HO80] High Order Language Working Group (HOLWG). *Requirements for Ada Programming Support Environments, STONEMAN*. U. S. Department of Defense, 1980.

[IEC95] Draft IEC Standard 1508 - Functional Safety: Safety-Related Systems, April 1995. International Electrotechnical Commission, Technical Committee no. 65, Working Group 9/10 (WG 9/10), IEC 65A.

[Irv01] C. Irvine et al. A Case Study in Security Requirements Engineering for a High Assurance System. In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[JMG+02] Trevor Jim, Greg Morrisett, Dan Grossman, Michael Hicks, James Cheney, and Yanling Wang. Cyclone: A safe dialect of C. In *USENIX Annual Technical Conference*, June 2002.

[KAJW93] S. Kumar, J. Aylor, B. Johnson, and W. Wulf. A Framework for Hardware/Software Codesign. *IEEE Computer*, 26(12):39–45, December 1993.

[KL94] A. Kalavadee and E. A. Lee. Manifestations of heterogeneity in hardware/software co-design. In M. Lorenzetti, editor, *Proceedings of the 31st Conference on Design Automation*, pages 437–438. ACM Press, June 1994.

[Knu94] D. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, May 1994. Also available in *Literate Programming*, D. Knuth (ed.), CSLI Lecture Notes 27, Center for the Study of Language and Information, 1992.

[Lee01] E. Lee. Overview of the Ptolemy Project. Technical Memorandum UCB/ERL M01/12, University of California, Berkeley, March 2001. `http://ptolemy.eecs.berkeley.edu/publications/papers/01/overview/`.

[Lev95] N. G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[LHR01] J. D. Lowrance, I. W. Harrison, and A. C. Rodriguez. Capturing Analytic Thought. In *Proc. of the First International Conference on Knowledge Capture*, pages 84–91, New York, October 2001. ACM Press.

[LLEL02] X. Liu, J. Liu, J. Eker, and E. Lee. Heterogeneous Modelling and Design of Control Systems. In *Software-Enabled Control: Information Technology for Dynamical Systems*, New York, NY, 2002.

[LRS+00] M. Lajolo, M. Rebaudengo, M. Sonza Reorda, M. Violante, and L. Lavagno. Evaluating System Dependability in a Co-Design Framework. In *Proceeedings of DATE'00*, Paris, France, March 2000.

[LSV98] L. Lavagno and A. Sangiovanni-Vincentelli. System-Level Design Models and Implementation Techniques. In *Proceedings of International Conference on Application of Concurrency to System Design*, 1998.

[LW97] Bev Littlewood and David Wright. A Bayesian model that combines disparate evidence for the quantitative assessment of system dependability. In Victoria Stavridou, editor, *Mathematics of Dependable Systems II*, pages 243–258. Oxford University Press, 1997.

[MA93] L. Lavango M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska and A. Sangiovanni-Vincentelli. A Formal Specification Model for Hardware/Software Codesign. In *Proc. International Workshop on Hardware-Software Codesign*, October 1993.

[MOD96] Safety Management Requirements for Defence Systems Containing Programmable Electronics. U.K. Ministry of Defence, August 1996. `http://www.dstan.mod.uk/`.

[MQRG97] M. Moriconi, X. Qian, R. Riemenschneider, and L. Gong. Secure Software Architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 84–93, Oakland, CA, May 1997.

[MXR95] M. Moriconi, Xiaolei Qian, and R. Riemenschneider. Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, 21(4):356–372, April 1995.

[Neu95] P. G. Neumann. *Computer Related Risks*. Addison-Wesley (ACM Press), 1995.

[NMW02] George C. Necula, Scott McPeak, and Westley Weimer. CCured: Type-safe retrofitting of legacy code. In *Principles of Programming Languages*. ACM, January 2002.

[NP99] P. G. Neumann and P. A. Porras. Experience with EMERALD to date. In *Workshop on Intrusion Detection and Network Monitoring*, April 1999.

[OB98] R. Ortega and G. Borriello. Communication Synthesis for Distributed Embedded Systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 437–444, San Jose, CA, November 1998.

[Olt01] K. Olthoff. Observations on Security Requirements Engineering. In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[Pal03] Partha Pal, editor. *DPASA Survivability Architecture: Final Design*. DARPA/AFRL, July 2003. Contract No. F30602-02-C-0134.

[PN97] P. Porras and P. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, MD, October 1997.

[Pur01] Purdue University Center for Education and Research in Information Assurance and Security (CERIAS). *Symposium on Requirements Engineering for Information Security*, March 2001.

[Rie99] R.A. Riemenschneider. Checking the Correctness of Architectural Transformation Steps via Proof-Carrying Architectures. In P. Donahoe, editor, *Software Architecture*. Kluwer Academic Press, 1999.

[Rus01] J. Rushby. Security Requirements Specifications: How and What? In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[San03] William H. Sanders, editor. *DPASA Critical Design Review Validation Report*. DARPA/AFRL, July 2003. Contract No. F30602-02-C-0134.

[Sch00] Fred B. Schneider. Enforceable security policies. *Information and System Security*, 3(1):30–50, 2000.

[Ste01] P. Stephenson. A Method for Security Requirements Engineering Using a Standards-Based Network Security Reference Model. In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[Sub93] P. Subrahmanyam. Hardware-Software Codesign: Cautious Optimism for the Future (Hot Topics). *IEEE Computer*, 26(1):84–85, 1993.

[SW01] K. Smith and P. Wu. Design of Future Navy Warships: A Method for the Development of Security Requirements within an Advanced Computing System. In *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, March 2001.

[VAC+01] Alfonso Valdes, Magnus Almgren, Steven Cheung, Yves Deswarte, Bruno Dutertre, Joshua Levy, Hassen Saïdi, Victoria Stavridou, and Tomás E. Uribe. An adaptive intrusion-tolerant server architecture. Technical report, System Design Laboratory, SRI International, CA, 2001.

[vDL98] A. van Lamweerde, R. Darimont, and E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, November 1998. Special Issue on Inconsistency Management in Software Development.

[vDM95] A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. In *Proceedings of the 2nd IEEE Symposium on Requirements Engineering*, pages 194–203, March 1995.

[WKM97] S. Wilson, T. Kelly, and J. McDermid. Safety Case Development: Current Practice, Future Prospects. In *Safety and Reliability of Software-Based Systems – Twelfth Annual CSR Workshop*, Bruges, Belgium, 1997.